Robust Congestion Control for ATM's ABR Service and Design of a Datalink Layer Interface Library

by

Mohammad Nazeeruddin

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

May, 1999

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

ROBUST CONGESTION CONTROL FOR ATM'S ABR SERVICE AND DESIGN OF A DATALINK LAYER INTERFACE LIBRARY

BY

MOHAMMAD NAZEERUDDIN

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

MAY 1999

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by

MOHAMMAD NAZEERUDDIN

under the direction of his Thesis Advisor, and approved by his Thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of

Master of Science in Systems Engineering

Thesis Committee :

Dr. O. Toker, **Chairman**

Dr. M. S. Ahmed, **Member**

Dr. F. Al-Sunni, **Member**

Dr. M. Bozyigit, **Member**

Dr. M. A. Al-Mouhamed, **Member**

Department Chairman

Dean of Graduate Studies

12/6/99

Date

DEDICATED AS A HUMBLE TRIBUTE TO

MY BELOVED PARENTS

WHOSE PRAYERS, SACRIFICE, INSPIRATION, AND LOVE

LED TO THIS ACCOMPLISHMENT

AND TO

MY LOVING BROTHER.

Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful.

Read in the name of thy Lord and Cherisher. Who created. Created man from a {leech-like } clot. Read and thy Lord is Most Bountiful. He Who taught {the use of} the pen. Taught man that which he knew not. Nay, but man doth transgress all bounds. In that he looketh upon himself as self-sufficient. Verily, to thy Lord is the return {of all}.

(The Holy Quran, Surah 96)

All praises are for ALLAH *subhanahu-wa-ta-Aaala*, the Most Compassionate, the Most Merciful. May peace and blessings be upon Prophet Muhammad, and his family. I thank Almighty Allah for giving me the knowledge and patience to complete this work. May He guide me and the whole humanity to the right path *(Aameen)*.

I acknowledge the support and facilities provided by the King Fahd University of Petroleum and Minerals for this work.

I would like to express my profound gratitude and appreciation to my thesis committee chairman Dr. Onur Toker, for his constant help, guidance and the attention that he devoted throughout the course of this work. He was always kind, understanding

and sympathetic to me.

Thanks are also due to my thesis committee members Dr. Mohammad Shahgir Ahmed, Dr. Fouad Al-Sunni, Dr. Muslim Bozyigit, and Dr. Mayez Al-Mouhamed for their interest, cooperation, advice and encouragement.

I also wish to thank the faculty and the staff members of the Systems Engineering Department for their support, especially to Dr. Abdul Basit Andijani for his moral support and cooperation.

Special acknowledgment is due to all my friends at KFUPM for their moral support, good wishes and the memorable days we shared together.

Finally, I thank my parents, brother, and grandmothers for their love, sacrifices, prayers and understanding. They helped me a lot in achieving my objectives.

Contents

List of Tables

List of Figures

Nomenclature

Notation and Symbols

$\Re$	Field of real numbers
A^H	Complex conjugate transpose of A
A'	Transpose of A
$Tr(A)$	Trace of A
A^{-1}	Inverse of A
$\|A\|_2$	H_2 norm of transfer function A
$\|A\|_\infty$	H_∞ norm of transfer function A
$\sigma_i(A)$	i^{th} singular value of A
$\rho(A)$	Spectral radius of A
$K(s)$	Transfer function matrix of the controller
$G(s)$	Transfer function matrix of the generalized plant
T_{zw}	Closed loop transfer function from w to z
u	Control inputs

w	External inputs
y	Measured outputs
z	Error signal
$Ric(H)$	The stabilizing solution of an ARE
$:=$	Defined as
$\in$	Belong to
$\cup$	Union
$\left[\begin{array}{c\|c} A & B \\ \hline C & D \end{array}\right]$	State space realization of $C(sI - A)^{-1}B + D$

Abbreviations

ARE	Algebraic Riccati Equation
ATM	Asynchronous Transfer Mode
ABR	Available Bit Rate
B-ISDN	Broadband Integrated Services Digital Network
BPF	BSD Packet Filter
BSD	Berkeley Software Distribution
CAC	Connection Admission Control
CBR	Constant Bit Rate
CCA	Congestion Control Algorithm

DLI	Datalink Layer Interface
DLPI	Datalink Layer Provider Interface
ER	Explicit Rate
LMI	Linear Matrix Inequality
MIMO	Multi Input Multi Output
QoS	Quality of Service
RM	Resource Management
SVR4	System V Release 4
TM 4.0	Traffic Management Version 4
UBR	Unspecified Bit Rate
UPC	Usage Parameter Control
VBR	Variable Bit Rate
VC	Virtual Circuit

THESIS ABSTRACT

Name: MOHAMMAD NAZEERUDDIN

Title: ROBUST CONGESTION CONTROL FOR ATM'S ABR SERVICE AND DESIGN OF A DATALINK LAYER INTERFACE LIBRARY

Major Field: SYSTEMS ENGINEERING

Date of Degree: May 1999

Congestion control is the most important part of ATM's ABR service since it has to fully utilize the available bandwidth without causing cell losses. The congestion control for high speed networks like ATM require a local flow controller at bottleneck nodes. In this thesis, we are using robust control theory to design explicit rate congestion controllers for ABR service. The controller design specifications considered here are fairness to multiple users, efficient utilization of the available bandwidth and minimization of queue size deviations. However, the most important design specification is the guaranteed performance in the presence of plant uncertainties and input disturbances. The performance of the designed controllers are evaluated by simulating a network under different practical conditions. The obtained results are quite encouraging. An user-level Datalink Layer Interface (DLI) library is also developed which is useful for testing and implementing congestion controller algorithms. This library is developed for both Berkeley Software Distribution (BSD) and System V Release 4 (SVR4) unices using object oriented features of C++.

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia.

May 1999.

خلاصة الرسالة

الاسم : محمد نذير الدين

العنوان : **التحكم المكين في الإزدحام لخدمة الـ ATMs ABR وتصميم مكتبة للتداخل بطبقة وصل المعلومات**

التخصص : هندسة النظم

تاريخ الشهادة : مايو ١٩٩٩م.

يعتبر التحكم في الإزدحام الجزء الأهم في خدمة الـ (ATMs ABR) إذ عليه أن يقوم بالاستغلال الكامل للحيز المتاح دون التسبب في فقدان خلايا. التحكم في الإزدحام لشبكات عالية السرعة مثل ATM يتطلب تحكماً موضعياً للتدفق لدى عقد الاختناق. تستخدم هذه الرسالة نظرية التحكم المكين لتصميم نظام للتحكم في الإزدحام جلي المعدل لخدمة ABR.

الوضوح بالنسبة للمستخدمين، الكفاءة في استغلال الحيز المتاح واختزال اختلافات حجم الصف هي ما أخذ في الاعتبار عند تصميم هذا النظام، على أن أهم مواصفات التصميم هو الأداء المضمون حال وجود المهمات والتشويش المصاحب للإدخال. تم تقويم أداء هذا النظام بمحاكاة شبكة تحت ظل ظروف عملية شتى وتم الحصول على نتائج مشجعة. أيضاً تم تطوير مكتبة للتداخل بطبقة وصلة المعلومات وهي تعتبر مفيدة في اختبار وتطبيق خوارزميات التحكم في الإزدحام. طورت هذه المكتبة لكل من النظامين (BSD) و (SVR4) وذلك باستخدام ملامح التوجه الموضوعي في لغة (C++).

ماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران – المملكة العربية السعودية

مايو ١٩٩٩ م

Chapter 1

INTRODUCTION

1.1 Introduction

This chapter gives an introduction to the main areas of the thesis. In Section 1.2, the readers are introduced to B-ISDN and ATM. In the later part of this section an overview of different service categories is given with more emphasis on ABR service class. In Section 1.3, an idea about congestion and congestion control algorithms is given. Datalink access is the focus of Section 1.4. In Section 1.5, motivation for this thesis work is given and in the final section an outline of the thesis is presented.

1.2 Introduction to B-ISDN and ATM

Information technology's growth has led to an obsession to provide better and more communication services than ever before. Computer communication networks are the latest manifestation of this obsession. Computer networks today primarily concerned with transferring data from one location to another without any real time constraints. However, there is technology available that allows simultaneous transmission of voice, video, graphics and text over the same network. Such networks are called Broadband Integrated Services Digital Networks (B-ISDNs). The B-ISDN is a digital virtual circuit for moving fixed size packets (Cells) from source to destination at 150 Mbits/sec.

A key technology for the B-ISDN is Asynchronous Transfer Mode (ATM). ATM is fundamentally a packet switching technology. The basic idea behind ATM is to transmit all information in small fixed size packets called Cells [1]. The Cells are 53 byte long packets, of which 5 bytes are for header and 48 bytes are for payload. There are variety of reasons for choosing the cell switching. Some of the important reasons are enumerated below [1].

1. Cell switching is highly flexible and can handle both constant rate traffic and variable rate traffic easily.

[1]This technology is called cell switching as an analogy to circuit switching where a copper path is established between source and destination within the telephone system.

2. At very high speed digital switching of the cells is easier than the traditional multiplexing techniques.

3. Broadcasting is essential for television distribution, which cell switching provides while, circuit switching cannot.

ATM networks are connection oriented. That is, it is required to first setup the connection before sending the data. The basic element of ATM is Virtual Circuit (VC). A VC is a connection from a source to a destination. VCs are unidirectional but a pair of circuits can be created at any time and both can be addressed by the same identifier. So, effectively a VC is full duplex.

In ATM cell delivery is not guaranteed, but the order is. The reason for this design is that ATM is designed for use on fiber optic network, which is highly reliable. Furthermore, ATM network are often used for real-time traffic. For this kind of traffic re-transmission of occasional bad cell is worse than ignoring it [2].

1.2.1 Service Categories

The ATM Forum completed the Traffic Management Version 4 (TM 4.0) specification [3]. The service categories specified by the ATM Forum are briefly described below.

- The **Constant Bit Rate (CBR)** class is intended to emulate a copper wire or optical fiber. In this service there is no error checking, flow control, or other

processing is done. Just the data put at one end and retrieved at the other end. However, this class is essential to make a smooth transition between the current telephone system and the future B-ISDN system, since voice grade Pulse Code Modulation (PCM) channels, T_1 circuits and most of the telephone system use constant rate synchronous bit transmission. CBR is also suited to all other real time audio and video streams.

- The **Variable Bit Rate (VBR)** is divided into the following two subclasses.

  1. Real Time-Variable bit rate (RT-VBR)
  2. Non Real Time-Variable bit rate (NRT-VBR)

The RT-VBR is intended for the services that have variable bit rates together with stringent real time requirements, such as video conferencing. In other words, the services which need both the average cell delay and the variation in cell delay to be strictly controlled are the best candidates for this service. The NRT-VBR class is for traffic where timely delivery is important but a

Table 1.1: The ATM service categories

Class	Description	Example
CBR	Constant bit rate	T_1 circuit emulation
RT-VBR	Variable bit rate: real time	Real-time video conferencing
NRT-VBR	Variable bit rate: non-real time	Multimedia e-mail
ABR	Available bit rate	Internet surfing
UBR	Unspecified bit rate	Background file transfer

certain amount of jitter can be tolerated by the application. For instance, multimedia e-mail is typically spooled to receiver's hard disk before being displayed.

- The **Available Bit Rate (ABR)** class is designed for bursty traffic whose bandwidth is known approximately. The ABR service avoids having to make a long term commitment to a fixed bandwidth. The ABR is the only service category in which the network provides rate feedback to the sender, asking it to slow down when the congestion occurs. Assuming that the sender complies with such requests, cell loss for the ABR traffic is expected to be low. The link bandwidth is first allocated to the CBR and VBR classes. The remaining bandwidth, if any, is given to ABR and UBR traffic.

- The **Unspecified Bit Rate (UBR)** service category makes no promises and gives no feedback about congestion to the sources. This category is well suited to send Internet Protocol (IP) packets, since IP also makes no promises about delivery. All UBR cells are accepted and delivered if there is some capacity left over.

The various service categories and their characteristics are summarized in Table 1.1 and Table 1.2 respectively.

1.2.2 ABR Traffic Management

The components of the ABR traffic management framework are shown in Figure 1.1. In order to obtain the network feedback, the sources send Resource Management (RM) cells after every n data cells. The destination simply returns these RM cells to the source. The RM cell contains an Explicit Rate (ER) field in which switches along the path indicate the rate at which the source should send after receipt of the RM cell.

One advantage of ER feedback is that each switch can calculate the rate at which it wants to allocate for the flow by its own method and reduce the ER field if necessary. The switches are not allowed to increase the field. There is no need to standardize a particular switch algorithm and TM 4.0 specification doesn't specify any standard algorithm [3]. While not requiring a switch algorithm is a feature of TM 4.0, the performance of the mechanism is heavily dependent on the switch algorithm. Some switch algorithms are very slow to respond to traffic changes, while others may be too fast. Some switch algorithms are unfair in certain circumstances, while others

Table 1.2: The characteristics of ATM service categories

Service Characteristic	CBR	RT-VBR	NRT-VBR	ABR	UBR
Bandwidth guarantee	Yes	Yes	Yes	Optional	No
Suitable for real-time traffic	Yes	Yes	No	No	No
Suitable for bursty traffic	No	No	Yes	Yes	Yes
Feedback about congestion	No	No	No	Yes	No

would be fair under those circumstances.

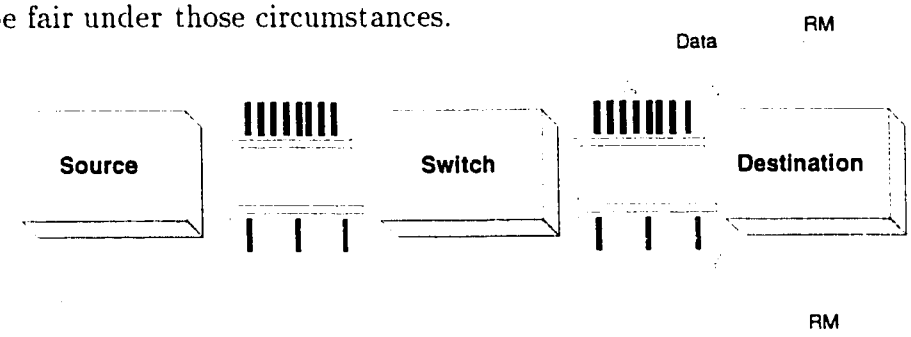


Figure 1.1: The ABR traffic management model: source, switch, destination and resource management cells

1.3 Introduction to Congestion Control

Data loss due to congestion is fundamental problem in ATM networks. This problem occurs when the sum of the bandwidth of the input streams crossing the node (switch) exceeds the output stream's bandwidth for an extended time period so that switch buffers fill up and overflow. Congestion results in poor end to end performance, which is especially very bad for ATM networks since loss of a single cell results in the re-transmission of the entire Protocol Data Unit (PDU), that is, a single dropped cell results in re-transmission of around 200 cells. Hence, it is necessary to reduce cell loss in ATM networks by using a proper congestion control algorithm.

1.3.1 Properties and Characteristics of Congestion Control Mechanisms

A good congestion control algorithm will have the following properties.

Fairness : The congestion control algorithm should allocate network resources in a fair manner among competing connections. Fairness has both static and dynamic components. The static fairness requirement is that, without changes in traffic mode the algorithm should converge to a state where competing sources share bandwidth fairly. Under dynamic conditions, short term unfairness is often unavoidable, but the system should quickly converge to a new fair state. Fairness requires a congestion control algorithm that both converges quickly and is responsive.

Application level performance : Applications typically evaluate network performance based on the time it takes to transfer their messages. Applications require low packet latency for small messages and high sustained throughput for large messages. A good congestion control algorithm performs well for a number of types of traffic, so that it can provide service to several types of applications. This requires flow control algorithm that is highly responsive and has low data loss.

Network utilization : Network resources must be utilized efficiently. This means that, even under dynamic network conditions, the congestion control mechanism should prevent under utilization of network links. Moreover, such an efficient link utilization should be achieved with minimal link resources on switches, for instance with small buffer sizes. This requires flow algorithm with low data loss, high responsiveness and low overhead.

To achieve the above properties the congestion control algorithm requires the following characteristics.

Data Loss : High data losses reduces application performance (decrease sustainable throughput and increases average latency) and network utilization. Congestion control algorithms should be designed to minimize data loss under severe congestion conditions and prevent congestion collapse of network.

Responsiveness : The responsiveness of the congestion control algorithms depends on the delay in the feedback loop. A good congestion control algorithm should be responsive to dynamically changing conditions in the network. The sources should be throttled back their data rate at the time of congestion and should be able to ramp up to any additional bandwidth that may become available. A poor responsiveness to traffic changes reduces application performance, fairness and network utilization.

Overhead : A good congestion control algorithm should have less overhead, that is, the network bandwidth requirements for control message is small and the processing requirements at the network nodes and end systems is low. High overhead reduces network utilization.

The congestion control algorithm features listed above are performance oriented. In addition, there are also a number of implementation oriented features. The important feature is, a good congestion control algorithm should be robust against loss of feedback information from the network. Second, the congestion control algorithm should be implementable. Finally, algorithm should be scalable, that is, they should perform well with varying speeds of links in the network and scale to large number of user connections.

1.4 Introduction to Datalink Layer Access

Datalink layer is the second layer of the Open Systems Interconnection (OSI) model. The datalink layer handles framing, addressing, error detection, collision avoidance and collision recovery. Framing is the process of creating a frame, also called a packet. A packet is a sequence of related bytes transmitted as a single logical unit. The packet's format specifies the frame used to transmit the data on the network.

The most of current workstation based operating systems are allowing user-level

applications to access the datalink layer directly. Under Unix there are two popular methods to access the datalink layer: BPF and DLPI. Datalink access provides the following important capabilities.

- It allows user level programs to watch the packets available on the network. If the network interface card has the capability to go into promiscuous mode, then all the packets on the local cable can be watched even though packets are not destined to the host on which the program is running.
- It allows certain programs to run as normal applications instead of as a part of kernel.

1.5 Motivation and Objective of Thesis

The exploding popularity of the Internet has fueled the demand to exchange the multimedia information over B-ISDN. The major component of B-ISDN is ATM technology, which has been designed to support the integration of voice, video and data applications. Thus, ATM networks are quickly being adopted as backbones at different parts of the Internet. While, multimedia applications are still in the development stage, most of the traffic on the Internet is data traffic. Hence data traffic should be properly managed.

Among the service classes provided by the ATM, UBR and ABR service classes are developed specifically to support the data applications. For UBR service switches

monitor their queues and simply discard the cells of overloading users. But, even a single cell-loss results in re-transmission of entire protocol data unit (around 200 cells). On the other hand, with ABR service ATM networks controls the congestion quickly and intelligently. Such an efficient control is potentially necessary on backbones where traffic is aggregate. *Moreover, it was shown that with good switch algorithm ABR pushes the congestion to the edge of the ATM* [4].

ABR service is not only recommended for data traffic but it is also proposed for transmission of Moving Pictures Experts Group (MPEG-2) [2] video [5, 6]. It is was shown that this approach can achieve high bandwidth utilization and guarantees continuous MPEG transmission by setting Minimum Cell Rate (MCR) to the bit rate under maximum compression ratio of encoder [5]. Therefore, with low prices MPEG-2 video can be transmitted over ABR. However, encoders need to be adaptive to control compression rates according to the feedback from the network.

ABR service is motivated to increase the bandwidth utilization by allocating residual (after allocating to CBR and VBR) bandwidth fairly among the applications. The available bandwidth (capacity) is subjected to sudden changes caused by arrival or departure of CBR or VBR connections and by short term congestion in an ATM switch caused by the simultaneous bursts from the several sources. Although ABR is a best

[2]The international standards organization about MPEG over ATM has adopted MPEG-2 transport stream format as the base-line of multimedia transmission.

effort service, the ATM forum has defined MCR to guarantee the services for ABR connections.

Congestion control is the most important part of ABR to fully utilize the available bandwidth without causing cell losses [7]. ATM forum has standardized rate-based approach as the framework of ABR congestion control because it has high flexibility and low hardware complexity [3, 8].

From the above paragraphs it should be evident that ABR is an important service of ATM and congestion control is the most important part of ABR. *Thus, in this thesis we have designed different explicit rate congestion controllers for ABR service and also developed a library which can be used to implement congestion controller algorithms.* Robust control theory is used in designing congestion controllers. The design specifications for these controllers include fairness to multiple users, efficient utilization of the available bandwidth and minimization of queue length deviations. However, the most important design objective is the robust performance in presence of the plant uncertainties and input disturbances. A network is simulated with the designed controllers and several experiments are conducted on this network to ensure controller's performance in a realistic environment. After designing the congestion controllers we have developed a datalink layer access library which is useful for testing and implementing controller algorithms. This library is developed using Object Oriented Programming (OOP) features of *C++* and supports both SVR4's

DLPI and BSD's BPF.

1.6 Organization of Thesis

This thesis is organized in six chapters. In this chapter, a brief introduction was given to the main areas of the thesis. In Chapter 2, a concise review of existing congestion control algorithms is presented and in Chapter 3, robust control theory related to designed controllers is reviewed. In Chapter 4, complete details of the conducted experiments and obtained results are presented and in Chapter 5 the development of the datalink layer interface library is described. Finally, in Chapter 6 some concluding remarks are made and future research directions are discussed.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

ATM networks must handle both long-term congestion, caused by more traffic coming in than the system can manage and short-term congestion, caused by burstiness in the traffic. As a result, a great deal of thought has gone into the subject of congestion control. In this chapter, we will discuss different approaches and strategies used in the literature to deal with congestion in the ATM networks.

2.2 Congestion Control Strategies

The overall congestion control strategies can be divided into the following categories:

1. Admission control

2. Resource reservation and usage parameter control

3. Rate-based congestion control

2.2.1 Admission Control

In low speed networks, it is usually adequate to wait for congestion to occur and then react to it by requesting the sources to slow down. But in high speed networks, due to long propagation delays, several thousands of additional packets may arrive before sources adjust their rates. Moreover, many ATM networks have real-time traffic sources that send data at an inherent rate. Asking such sources to slow down may not work. Furthermore, there is no dynamic control present for CBR, VBR and UBR traffic. Therefore, it is desirable to use preventive control mechanisms. Admission control is the most important preventive control mechanism. It decides whether or not to accept a new VC based on knowledge of the current traffic load, the new VC's QoS requirements, etc. Denying admission should be done fairly. Otherwise a small number of high bandwidth users can adversely affect many low bandwidth users.

A Connection Admission Control (CAC) scheme should satisfy the following three requirements.

- It should be fast enough to make real time decisions.

- It should support multiple ATM service classes.
- It should be simple and cost effective.

A real time CAC proposed in [9] is suitable for an environment in which multiple service categories are available. It was based on a cell loss ratio evaluation algorithm and it can support CBR, NRT-VBR, RT-VBR, ABR and UBR. However, ABR is restricted in such a way that MCR must be zero. But, real time services on the data networks are currently increasing and hence there is need for a CAC, which supports ABR with a non-zero MCR.

In [10] such a CAC scheme was proposed. This scheme is based on two featured technologies. One is a real time cell loss ratio evaluation algorithm for VBR. The other is a real time evaluation of available bandwidth. The important features of this scheme are that it terminates in a few milliseconds and this time is independent of the number of VC's. However, these algorithms assume that exact traffic characteristics are available and this is not true since users may not have exact characteristics of their traffic before transmitting it.

In [11] a new neuro computing based CAC algorithm is proposed. This algorithm employs NN's to calculate the bandwidth required per call using on-line measurements of the traffic via its count process, instead of relying on simple parameters such as peak bit rates, average bit rates and burst length.

2.2.2 Resource Reservation and Usage Parameter Control

Related to the CAC is the technique of reserving the resources in advance, usually at the call setup time. As the traffic descriptor contains peak and average cell rates, the network has the possibility of reserving enough band width along the path to handle the rates. The Usage Parameter Control (UPC) (i.e., policing or traffic enforcement) is a mechanism that ensures that a VC/VP does not exceed its agreements with the network, that is, that it does not exceed its traffic descriptor. The main objectives of UPC are to detect any violations in the traffic parameters and to react to these violations by taking an appropriate action that will police the traffic back to its non-violating pattern. The UPC could take any of the following actions.

Cell discard : The UPC discards excessive traffic at the entrance of the network itself. This scheme can be realized through the use of a *leaky bucket* scheme [15] that monitors and controls the peak and mean cell rates of the VC/VP.

Violation tagging : The UPC tags the excessive traffic of a VC/VP as low priority traffic. In the event of congestion, the network discards the excessive cells [16]. The advantage of this method is that it encourages the users to utilize the network and may be useful for services that do not need a stringent QoS. But, when the network guarantees the QoS for a VC, this scheme may become ineffective due to the fact that the network could not discard the

marked cells of different VCs fairly [13].

Quenching the source : The UPC instructs the source to adjust its rate when a VC/VP violates its traffic descriptors. Any of the available flow control algorithms can be used for quenching the sources [12].

Most of the existing algorithms attempt to police the peak and average bit rates of the traffic. However, this approach checks only one parameter of the probability density function (pdf) of the traffic. Thus, violations could pass undetected by the algorithm. Moreover, the mean bit rate cannot be estimated accurately. Hence, the algorithm will be attempting to detect violations with an already inaccurately measured parameter. In [14] a policing mechanism using neural networks is proposed. It is based on an estimation of the probability density function (pdf) of the traffic via a count process. This mechanism does not explicitly calculate the pdf of the traffic, but achieves via a neural network which is trained to learn the pdf of the traffic's count process through many training data.

2.2.3 Rate-Based Congestion Control

In a rate-based congestion control scheme, sources are requested to slow down at the time of congestion. This kind of congestion control is reasonable and possible with only ABR traffic. Since for CBR and VBR traffic, it is generally not possible to slow down due to the real time or semi-real time nature of the sources and for UBR, due

to lack of congestion control scheme. There are many proposed solutions available in the literature [8, 27, 28, 29, 30] to detect, signal and control the congestion in the ABR traffic. Lets examine some of the solutions with their short comings and advantages.

One solution suggested that whenever a sender wishes to send a burst of data, it had to send a special cell for reserving the necessary bandwidth. The advantage here is that congestion will never occur since the required bandwidth is always reserved before sending the data. But ATM forum has rejected this proposal due to the chance of long delay before a host may begin to send actual data.

Some other solutions had switches sending *back choke cells* whenever congestion began to occur. Whenever a source receives such a cell it reduces its cell transmission rate. Several ways are proposed for getting the rate back up again later when congestion is cleared. These solutions are also rejected because *back choke cells* might get lost in the congestion and also because these solutions seen to be unfair to the small users.

Some proposals used the fact that packet boundaries are marked by a bit in the last cell. The idea is to discard the cells to relieve the congestion but to do this highly selectively. The switch has to scan the input stream and discard cells those belong to the same packet (which would ultimately result in re-transmission of a single packet) instead of dropping random cells (which might result in re-transmission of several packets). However, this scheme was also rejected due to the fact that discarded cells

might not belong to the overloading source. Hence, this scheme might not be fair. Another proposal [18] suggested a credit based solution, which is essentially a dynamic sliding window protocol. This solution requires each switch to maintain a credit (number of buffers reserved) per VC. Thus, congestion will never occur provided there is a buffer waiting for each transmitted cell. This scheme is also rejected since the switch has to do a lot of accounting to keep track of credits and many buffers have to be reserved in advance. This may lead to unnecessary overhead and wastage.

After much discussion, the ATM Forum has adopted the rate-based congestion scheme [17]. In this scheme a source periodically sends a RM cell. The switch notifies its congestion by an Explicit Forward Congestion Indication (EFCI) bit of data cells or a Congestion Indication (CI) bit of RM cells. Since the switch uses a single bit to inform the congestion it is generally referred to as **binary mode** switch. In the ATM standard [3], the switch is allowed to communicate explicitly the cell transmission rate by modifying the ER value of the RM cell. This type of switch is called an **explicit rate** switch.

Explicit Rate Congestion Control

The ER switch has the potential to achieve much better performance than the binary mode switch. The normal operation of the ER switch is to compute an appropriate rate for every connected source based on available bandwidth (C) for ABR connec-

tions, the degree of congestion, etc. The switch then modifies the ER value of RM cells. When the sources receive the RM cell, they updates their Allowed Cell Rate (ACR).

Several switch algorithms with ER marking have been proposed through the standardization process of rate based algorithms. Some of them are the Explicit Proportional Rate Control Algorithm (EPRCA) [19], Congestion Avoidance using Proportional rate Control (CAPC) [20], Adaptive Proportional Rate Control (APRC2) [21], Explicit Rate Indication Congestion Avoidance (ERICA) [22] and the Max-Min scheme [23]. Each algorithm has its own advantages and disadvantages in terms of effectiveness, fairness, robustness, etc.

The congestion control problem is also looked at from the control engineering viewpoint [24]. In [25] the rate based congestion control problem was examined using control theory. A congestion control scheme for simple networks was designed and analyzed using the tools of classical control theory. They were able to predict the performance of the algorithms analytically rather than relying on simulations. In [26], a H_∞ based controller for rate feedback in a single bottleneck network was proposed. By simple algebra the problem is transformed to a H_∞ controller of a plant with a time delay and it is solved using an algorithm developed for this class of problems.

Chapter 3

REVIEW OF ROBUST CONTROL THEORY

3.1 Introduction

In this chapter a brief review of theory related to the controllers used in this thesis is given. The derivations of the results presented in this chapter are skipped and for further details readers are directed to [31, 35, 34, 37]. The material covered is more or less sufficient to understand the subsequent chapters. H_2, H_∞, mixed H_2/H_∞ and adaptive controls theory is covered respectively in Sections 3.2, 3.3, 3.4 and 3.5.

3.2 Standard H_2 Problem

The system considered in this section is described by the standard block diagram shown in Figure 3.1, where G is the generalized plant and K is the controller. The generalized plant G contains what is usually called a plant in a control problem plus all weighting functions. The signal w contains all the external inputs, including disturbances, sensor noise and commands; the output z_2 is an error signal; y is the measured variables; and u is the control input. The diagram in Figure 3.1 is also referred to as a Linear Fractional Transformation (LFT) on K, and G is called the coefficient matrix for the LFT. The resulting closed loop transfer function from w to z_2 is denoted by T_{z_2w}.

The realization of the transfer matrix G is taken to be of the form

$$G(s) = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & 0 \end{array}\right] \tag{3.1}$$

Notice the special structure of D: D_{22} is assumed to be zero so that G_{22} is strictly proper. Note that this assumption is made without loss of generality since a substitution of $K_D = K((I + D_{22}K^{-1}))$ would give controller for $D_{22} \neq 0$. The following additional assumptions are made [31].

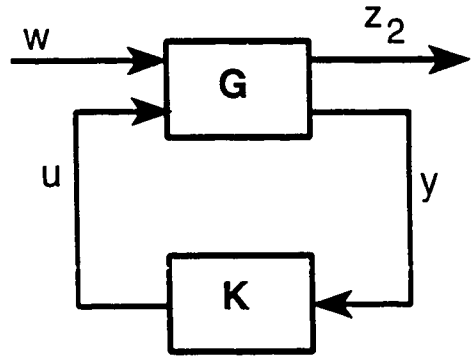


Figure 3.1: The Block Diagram of H_2 Controller

1. $(A,\ B_2)$ is stabilizable and $(C_2,\ A)$ is detectable.

2. $R_1 = D_{12}^{II} D_{12} > 0$ and $R_2 = D_{21} D_{21}^{II} > 0$.

3. The matrix

$$\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix} \tag{3.2}$$

has full column rank for all ω.

4. The matrix

$$\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix} \tag{3.3}$$

has full row rank for all ω.

Assumption 1 is necessary for the existence of stabilizing controllers. *Assumption 2* guaranties that the H_2 optimal control problem is non-singular. *Assumptions 3 and 4* are technical to guarantee the existence of solutions to certain Riccati equations.

Then the H_2 control problem is to find a proper, real rational controller K such that K stabilizes the generalized plant G internally and minimizes the H_2 norm of the transfer function matrix T_{zw} from w to z_2.

3.2.1 The H_2 Norm of Continuous Time Systems

Consider a continuous time system having a q x l stable transfer function $G(s)$. Then the H_2 norm of $G(s)$ is defined as

$$\|G\|_2 = \left(\frac{1}{2\pi}\mathrm{Tr}\left[\int_{-\infty}^{+\infty} G(j\omega)G^H(j\omega)d\omega\right]\right)^{1/2}. \tag{3.4}$$

By Parseval theorem, $\|G\|_2$ can equivalently be defined as

$$\|G\|_2 = \left(\mathrm{Tr}\left[\int_{0}^{+\infty} g(t)g^H(t)dt\right]\right)^{1/2}, \tag{3.5}$$

where $g(t)$ is the unit impulse (Dirac distribution) response matrix of $G(s)$. Thus, $\|G\|_2 = \|g\|_2$. The H_2 norm of $G(s)$ can also be expressed in terms of the singular values of the matrix $G(j\omega)$.

$$\|G\|_2 = \left(\frac{1}{2\pi}\int_{-\infty}^{+\infty}\left[\sum_{i=1}^{min\{q,\,l\}}\sigma_i^2(G(j\omega))d\omega\right]\right)^{1/2} \tag{3.6}$$

where $\sigma_i(G(j\omega))$ is the i^{th} singular value of $G(j\omega)$.

Stochastic interpretation of H_2 norm

Let us consider a system with a stable transfer function $G(s)$. Let the input $w(t)$ be a wide-sense stationary stochastic process. Let $z(t)$ be the corresponding output. It is well known that

$$S_z(\omega) = G(j\omega)S_w(\omega)G^H(j\omega) \tag{3.7}$$

where $S_z(\omega)$ and $S_w(\omega)$ are the power spectral densities of $z(t)$ and $w(t)$ respectively. Then, the H_2 norm of $G(s)$ *can be interpreted as the energy of the output* $z(t)$ *when the given system is driven by independent zero mean white noise with unit power spectral density.*

A state-space method for computing the H_2 norm

Consider a continuous time transfer function $G(s)$ described in packed notation,

$$G(s) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{3.8}$$

with A being Hurwitz-stable. Let W_{obs}^c denotes the observability grammian of the pair $(A,\ C)$, and W_{con}^c denotes the controllability grammian of the pair $(A,\ B)$ [34]. Then for $D = 0$, W_{obs}^c and W_{con}^c are the solutions of the continuous time Lyapunov equations,

$$A'W_{obs}^c + W_{obs}^c A + C'C = 0 \text{ and } AW_{con}^c + W_{con}^c A' + BB' = 0 \tag{3.9}$$

H_2 norm of $G(s)$ can now be computed by

$$\|G(s)\|_2 = \text{Tr}\,[B'W_{obs}^c B] = \text{Tr}\,[CW_{con}^c C'] \tag{3.10}$$

3.2.2 The H_2 Norm of Discrete-Time Systems

Consider a discrete time system having a $q \times l$ stable transfer function $G(s)$. Then the H_2 norm of $G(z)$ is defined as

$$\|G\|_2 = \left(\frac{1}{2\pi}\text{Tr}\left[\int_{-\pi}^{+\pi} G(e^{j\omega})G^H(e^{j\omega})d\omega\right]\right)^{1/2}. \tag{3.11}$$

Again, by the Parseval theorem, $\|G\|_2$ can equivalently be defined as

$$\|G\|_2 = \left(\text{Tr}\left[\sum_{k=0}^{\infty} g(k)g^H(k)dt\right]\right)^{1/2}. \tag{3.12}$$

where $g(t)$ is the impulse response matrix of $G(z)$. Thus, as in the continuous case, $\|G\|_2 = \|g\|_2$.

Stochastic interpretation of H_2 norm

As in the continuous-time case, the H_2 norm of $G(z)$ *can be interpreted as the energy of the output* $z(k)$ *when the given system is driven by independent zero mean white noise sequences having unit variances.*

A state-space method for computing H_2 norm

Consider a discrete-time transfer function $g(z)$ described in packed notation,

$$G(z) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{3.13}$$

with A being Schur-stable. Let W^d_{obs} denote the observability grammian of the pair $(A,\ C)$, and W^d_{con} denote the controllability grammian of the pair $(A,\ B)$. Then for $D = 0$, W^c_{obs} and W^c_{con} are the solutions of discrete time Lyapunov equations,

$$A'W^d_{obs} + W^d_{obs}A + C'C = 0 \text{ and } AW^d_{con} + W^d_{con}A' + BB' = 0 \tag{3.14}$$

The H_2 norm of $G(z)$ can now be computed by

$$\|G(z)\|_2 = \mathrm{Tr}\left[B'W^d_{obs}B\right] = \mathrm{Tr}\left[CW^d_{con}C'\right] \tag{3.15}$$

3.2.3 Solution to the Continuous Time Systems

The solution to the H_2 problem under above stated assumptions can be solved using the Riccati based approach [31]. However, there is yet another method based on Linear Matrix Inequalities (LMI) [32], which does not need any of the above assumptions. But the LMI-based approach is computationally more involved. In this section the final equations to be solved to get (sub) optimal controllers for both methods are given directly. For further details see [34].

Riccati-based approach

The problem is to find an admissible controller K which minimizes $\|T_{zw}\|_2$. Consider the following two Hamiltonian matrices

$$H_\infty := \begin{bmatrix} A & -\gamma^{-2}B_1B_1^H - B_2B_2^H \\ -C_1^HC_1 & -A^H \end{bmatrix} \tag{3.16}$$

$$J_\infty := \begin{bmatrix} A^H & -\gamma^{-2}C_1^HC_1 - C_2^HC_2 \\ -B_1B_1^H & -A \end{bmatrix} \tag{3.17}$$

belong to $dom(Ric)$ [31], and, moreover, $X_2 := Ric(H_2) \geq 0$ and $Y_2 := Ric(J_2) \geq 0$. Now define

$$F_2 := -(B_2^HX_2 + D_{12}^HC_1),\ L_2 := -(Y_2C_2^H + B_1D_{21}^H) \tag{3.18}$$

and

$$A_{F_2} := A + B_2F_2, \quad C_{1F_2} := C_1 + D_{12}F_2 \tag{3.19}$$

$$A_{L_2} := A + L_2C_2, \quad B_{1L_2} := B_1 + L_2D_{21} \tag{3.20}$$

$$\hat{A}_2 := A + B_2F_2 + L_2C_2 \tag{3.21}$$

$$G_c(s) := \left[\begin{array}{c|c} A_{F_2} & I \\ \hline C_{1F_2} & 0 \end{array}\right], \qquad G_f(s) := \left[\begin{array}{c|c} A_{L_2} & B_{1L_2} \\ \hline I & 0 \end{array}\right]. \tag{3.22}$$

Then there exists a unique optimal controller given by

$$K_{opt}(s) := \left[\begin{array}{c|c} \hat{A}_2 & -L_2 \\ \hline F_2 & 0 \end{array}\right]. \tag{3.23}$$

Moreover, $\min \|T_{zw}\|_2^2 = \|G_cB_1\|_2^2 + \|F_2G_f\|_2^2 = \|G_cL_2\|_2^2 + \|C_1G_f\|_2^2$

The family of all admissible controllers such that $\|T_{zw}\|_2 < \gamma$ is the set of all transfer matrices from y to u in Figure 3.2.3.

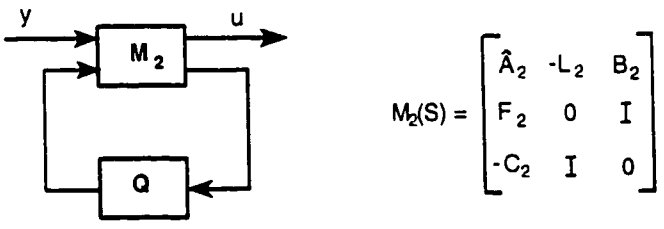


Figure 3.2: Family of admissible controllers

where $Q \in \Re H_2$, $\|Q\|_2^2 < \gamma^2 - \|G_c B_1\|_2^2 + \|F_2 G_f\|_2^2$

Thus, the suboptimal controllers are parameterized by a fixed (independent γ) linear fractional transformation with a free parameter Q. With $Q = 0$, we recover K_{opt}.

LMI-based approach

Let

$$\begin{pmatrix} \dot{x} = Ax + B_1 w + B_2 u \\ z_2 = C_2 x + D_{21} w + D_{22} u \\ y = C_y x + D_{y1} w \end{pmatrix}, \tag{3.24}$$

$$\begin{pmatrix} \dot{x}_{cl} = A_{cl} x_{cl} + B_{cl} w \\ z_2 = C_{cl2} x_{cl} + D_{cl2} w \end{pmatrix} \tag{3.25}$$

be the state-space realizations of the plant P and the corresponding closed-loop system respectively. The closed loop H_2 norm doesn't exceed γ if and only if $D_{cl2} = 0$ and there exist two symmetric matrices X_2 and Q such that the following LMI formulation is satisfied.

$$\text{Minimize} \quad Tr(Q)$$

satisfying the following constraints

$$\begin{pmatrix} A_{cl}X_2 + X_2A_{cl}' & B_{cl} \\ B_{cl}' & -I \end{pmatrix} < 0,$$

$$\begin{pmatrix} Q & C_{cl2}X_2 \\ X_2C_{cl2}' & X_2 \end{pmatrix} > 0,$$

$$Tr(Q) < \gamma^2.$$

Given optimal solution Q^* the closed loop H_2 performance is bounded by $\|T_{zw}\|_2 \leq \sqrt{(Tr(Q^*)}$ [33].

3.3 Standard H_∞ Problem

The standard block diagram of the H_∞ controller is shown in Figure 3.3, where the plant G and controller K are assumed to be real-rational and proper. It is assumed that state space models of G and K are available and that realization is assumed to be stabilizable and detectable. Then *the H_∞ control problem is to find all admissible controllers $K(s)$ such that the H_∞ norm of the transfer function from w to z_{inf} ($\|T_{zw}\|_\infty$) is minimized.*

A controller is said to be admissible if it internally stabilizes the system. It should be noted that the optimal H_∞ controllers as defined above are generally not unique

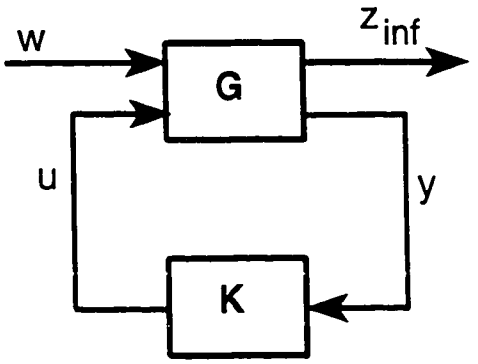


Figure 3.3: The Block diagram of H_∞ controller

for MIMO systems. Furthermore, in contrast to the standard H_2 problem, finding an optimal H_∞ controller is often both numerically and theoretically complicated [35]. Theoretically knowing the achievable H_∞ norm may be useful since it sets a limit on what we can achieve. But, in practice it is often not necessary and sometimes even undesirable to design an optimal controller, and it is much cheaper to obtain suboptimal controllers that are very close in performance to the optimal ones. A *suboptimal H_∞ control problem is to find all admissible controllers K for a given $\gamma > 0$, if there are any, so that* $\| T_{zw} \|_\infty < \gamma$

The realization of G and the assumptions made about G are the same as those defined in the Section 3.2.

3.3.1 The H_∞ Norm

Consider a continuous time system having a q x l stable transfer function $G(s)$. Then the H_∞ norm of $G(s)$ is defined as

$$\|G\|_\infty := \sup_\omega \sigma_{max}[G(j\omega)]. \tag{3.26}$$

Further, let $w(t)$ and $z(t)$ be the input and the corresponding output energy signals of the system $G(s)$. Then $\|G\|_\infty$ can also be interpreted as

$$\|G\|_\infty = \sup_{\|w\|_2 \neq 0} \frac{\|z\|_2}{\|w\|_2}. \tag{3.27}$$

Similarly, consider a discrete time system having a $q \times l$ stable transfer function $G(z)$. Then the H_∞ norm of $G(z)$ is defined as

$$\|G\|_\infty := \sup_{-\pi \leq \omega \leq \pi} \sigma_{max}[G(e^{j\omega})]. \tag{3.28}$$

Equation (3.27) is also valid for any discrete system $G(z)$ with $w(k)$ and $z(k)$ as the input and the corresponding output energy signals respectively.

An important property of the H_∞ norm, for both continuous and discrete systems, is that it is submultiplicative. That is, for the transfer matrices G_1 and G_2, we have

$$\|G_1 G_2\|_\infty \leq \|G_1\|_\infty \|G_2\|_\infty \tag{3.29}$$

Interpretation of H_∞ norm

Consider a proper and stable transfer function matrix $H(s)$, that is, $H(s) \in \Re H_\infty$. Suppose a $\Delta(s) \in \Re H_\infty$ is connected from the output of $H(s)$ to the input of $H(s)$ as shown in Figure 3.4. Then the closed loop system is internally stable if

$$\sigma_{max}[\Delta(j\omega)]\sigma_{max}[H(j\omega)] < 1 \quad \forall\, \omega \in \Re \cup \{\infty\}. \tag{3.30}$$

This is called the Small-Gain Theorem. In the view of the Small-Gain Theorem, if we

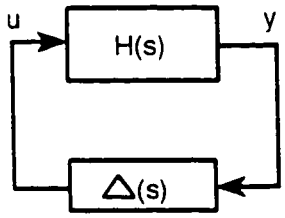


Figure 3.4: A certain closed-loop system

have $\|H(s)\|_\infty < \gamma$, then $\Delta(s)$ being an element of $\Re H_\infty$ together with $\|H(s)\|_\infty < \gamma^{-1}$ is sufficient to guarantee the internal stability of the closed-loop system given in Figure 3.4. If we now interpret $\Delta(s)$ as a perturbation of the given system (representing uncertainty), then the small-gain theorem tells us that an H_∞ norm bound on $H(s)$ implies stability in the presence of certain H_∞ norm bounded system uncertainties. In other words, the H_∞ norm bound implies a pre-specified level of stability robustness. The robustness increases as the H_∞ norm bound on $H(s)$ decreases, that is, the size of tolerable uncertainty varies inversely proportional to

γ. A similar discussion is applicable for discrete-time systems as well.

State-space method for computing H_∞ norm

Regarding the H_∞ norm computation, there is a simple way to determine whether the inequality specification $\|G\|_\infty < \gamma$ is satisfied. For $G(s)$ in Equation (3.8), with $\gamma > 0$, we define the matrix

$$M_\gamma = \begin{bmatrix} A + BR^{-1}D^H C & \gamma^{-2}BR^{-1}B^H \\ -C^H(I + DR^{-1}D^H)C & -(A + BR{-}1D^H C)^H \end{bmatrix} \tag{3.31}$$

where $R := \gamma^2 I - D^H D > 0$. Then, we have $\|G\|_\infty < \gamma \iff M_\gamma$ has no imaginary eigen values, and the maximum singular value of $D < \gamma$. Thus, to determine $\|G\|_\infty$ numerically, one can select a positive number γ, determine if $\|G\|_\infty < \gamma$ by calculating the eigen values of M_γ, and increase or decrease γ accordingly until the desired precision is reached. The computation of the H_∞ norm of a discrete-time system's transfer matrix follows a similar procedure.

3.3.2 Solution to the Continuous Time Systems

In this section two different methods to synthesize the H_∞ controller are presented. They are

1. the Riccati-based approach, and

2. the LMI-based approach.

Recall that the LMI-based approach does not impose any assumptions on the plant ($G(s)$). However, it is computationally more involved than the Riccati-based method. Again, in this section the final equations to be solved to get sub-optimal H_∞ controllers are given directly. For further details about these results see [36].

Riccati-based approach

The problem is to find all admissible K so that $\|T_{zw}\|_\infty < \gamma$. Clearly, γ must be greater that the optimal H_∞ level. Optimal H_∞ controllers are more difficult to synthesize than suboptimal ones. This is the major difference between H_2 and H_∞ results. A similar difference arises in the norm computation. The H_∞ solution involves two new Hamiltonian matrices

$$H_\infty := \begin{bmatrix} A & -\gamma^{-2}B_1B_1^H - B_2B_2^H \\ -C_1^HC_1 & -A^H \end{bmatrix}. \tag{3.32}$$

$$J_\infty := \begin{bmatrix} A^H & -\gamma^{-2}C_1^HC_1 - C_2^HC_2 \\ -B_1B_1^H & -A \end{bmatrix}. \tag{3.33}$$

The important difference here is that we cannot guarantee that $H_\infty \in dom(Ric)$ or $Ric(H_\infty) \geq 0$. Indeed, these conditions are intimately related to the existence of H_∞ suboptimal controllers.

There exists an admissible controller such that $\|T_{zw}\|_\infty < \gamma$ if and only if the following three conditions are satisfied

1. $H_\infty \in dom(Ric)$ and $X_\infty := Ric(H_\infty) \geq 0$.

2. $J_\infty \in dom(Ric)$ and $Y_\infty := Ric(J_\infty) \geq 0$.

3. $\rho(X_\infty Y_\infty) < \gamma^2$.

Moreover, when these conditions hold, one such controller is

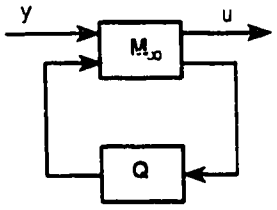
$$K_{sub}(s) := \left[\begin{array}{c|c} \hat{A}_\infty & -Z_\infty L_\infty \\ \hline F_\infty & 0 \end{array}\right]. \tag{3.34}$$

where

$$\hat{A}_\infty := A + \gamma^{-2} B_1 B_1^H X_\infty + B_2 F_\infty + Z_\infty L_\infty C_2. \tag{3.35}$$

$$F_\infty := -B_2^H X_\infty, \; L_\infty := -Y_\infty C_2^H, \; Z_\infty := (I - \gamma^{-2} Y_\infty X_\infty)^{-1}. \tag{3.36}$$

If the above three conditions are satisfied, then the set of all admissible controllers such that $\|T_{zw}\|_\infty < \gamma$ equals the set of all transfer matrices from y to u in Figure 3.3.2



$$M_\infty(S) = \left[\begin{array}{ccc} \hat{A}_\infty & -_\infty L_\infty & Z_\infty B_2 \\ F_\infty & 0 & I \\ -C_2 & I & 0 \end{array}\right]$$

where $Q \in \Re H_\infty$, $\|Q\|_\infty < \gamma$.

Thus, as in the H_2 case, the suboptimal controllers are parameterized by a fixed linear fractional transformation with a free parameter Q. With $Q = 0$, we recover $K_{sub}(s)$.

LMI-based approach

Let

$$\begin{pmatrix} \dot{x} = Ax + B_1 w + B_2 u \\ z_\infty = C_\infty x + D_{\infty 1} w + D_{\infty 2} u \\ y = C_y x + D_{y1} w \end{pmatrix}, \tag{3.37}$$

$$\begin{pmatrix} \dot{x}_{cl} = A_{cl} x_{cl} + B_{cl} w \\ z_\infty = C_{cl1} x_{cl} + D_{cl1} w \end{pmatrix} \tag{3.38}$$

be the state-space realizations of plant P and the corresponding closed-loop system respectively.

The closed loop H_∞-norm does not exceed γ if and only if there exists a symmetric matrix X_∞ such that the following LMI formulation should be satisfied

Minimize γ satisfying:

$$\begin{pmatrix} A_{cl} X_\infty + X_\infty A'_{cl} & B_{cl} & X_\infty C'_{cl} \\ B'_{cl} & -I & D'_{cl1} \\ C_{cl1} X_\infty & D_{cl1} & -\gamma^2 I \end{pmatrix} < 0,$$

$$X_\infty > 0.$$

Given optimal solution γ^*, the closed loop H_∞ performance is bounded by $\|T_{zw}\|_\infty \leq \gamma^*$ [33].

3.4 Mixed H_2/H_∞ Controller

In many real world applications, the standard H_∞ system cannot adequately capture all design specifications, for example noise attenuation or regulation against random disturbances. But these specifications can be easily expressed in H_2 terms. On the other hand, H_2 solutions cannot provide global system independent guaranteed robustness properties. Therefore, to have the advantages of both H_2 and H_∞ controllers, a mixed H_2/H_∞ controller is designed.

The standard block diagram of the H_2/H_∞ system is shown in Figure 3.5. The output channel with the z_{inf} is associated with the H_∞ performance while the channel z_2 is associated with the H_2 performance. Let T_∞ and T_2 denote the closed loop transfer functions from w to z_{inf} and z_2 respectively. Then, the mixed H_2/H_∞ problem is to design an output feedback controller such that it minimizes a trade-off criterion of the form

$$a \parallel T_\infty \parallel_\infty^2 + b \parallel T_2 \parallel_2^2 \tag{3.39}$$

subjected to,

$\| T_\infty \|_\infty < \gamma$ and

$\| T_2 \|_2 < \beta$,

where a, b, γ and β are positive constants.

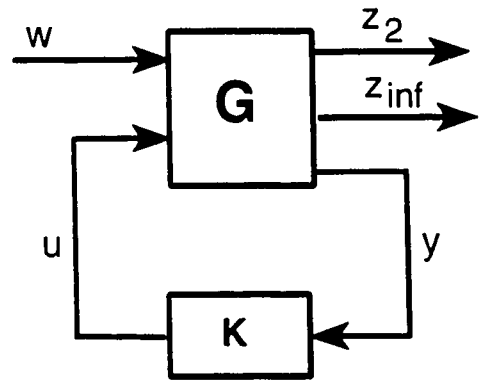


Figure 3.5: The Block Diagram of mixed H_2/H_∞ Controller

3.4.1 Solution to the Continuous Time Systems

The formulation of mixed H_2/H_∞ problem is more involved and it is skipped here. For a Riccati based formulation a reader can refer to [31] and for a LMI based formulation a reader can see [33, 38].

3.5 Adaptive Controller

An adaptive controller is a controller that can modify its behavior in response to the changes in the dynamics of the process and the disturbances. Since ordinary non adaptive controllers has been introduced for the same purpose then immediate question that needs to be answered is: *what is the difference between feedback control and adaptive control* ? Over the years there have been many attempts to define adaptive control. However, a meaningful definition of adaptive control is still lacking. In general adaptive control is a special type of non-linear feedback control in which states of the process can be separated into two categories, which change at different rates. The slowly changing states are viewed as parameters. This introduces the idea of 2-time scales: a fast time scale for ordinary feedback and a slow time scale for updating regulator parameters. Adaptive control is not a matured field. Many of the algorithms and approaches used are of an ad hoc nature. The tools are gathered from a wide range of fields and good systematic approaches are still lacking. Still these algorithms have good uses and there are adaptive systems that clearly outperform conventional feedback systems.

In the following subsection a heuristic based adaptive technique used in this thesis is explained.

3.5.1 A Heuristic Based Adaptive Technique

Consider a general feedback system like the one shown in the Figure 3.3. Let the controller be implemented by the equation

$$I(n) = k_1 + k_2[Q(n-1) - Q(n)] + k_3[C(n) - C(n-1)]. \qquad (3.40)$$

Where $I(n)$ is the maximum allowable rate to the sources at the time interval n, $Q(n)$ and $C(n)$ are the queue length and the available capacity at the sampling time n, and k_1, $k_2$and k_3 are the controller parameters.

Initially when the queue is empty, if the available capacity increases by C then the maximum allowable rate in the Eq. 3.40 reduces to $k_1 + k_3C$. Moreover, if $k_1 = 0$, then I is given by k_3C. If there are N number of sources, the total incoming rate is NI and ideally this rate should be equal to the outgoing rate (available capacity C). This means, $Nk_3C = C$ or $k_3 = \frac{1}{N}$. In the same way, when there is no change in the available capacity and if $k_1 = 0$ then k_2 can be approximated as $\frac{1}{N}$.

In the steady state, there will be no change in the queue lengths and available capacities. Thus, the controller should continue giving the same feedback to the sources. This means, $I(n)$ should be equal to $I(n-1)$. But from the Eq. (3.40), $I(n) = k_1$. Hence, $k_1 = I(n-1)$.

Now we have approximate values for k_1, k_2 and k_3. However, the above analysis is for a static case. But, the network is subjected to disturbances and state of the network changes dynamically. Thus, one or more parameters of the controller should be varied to adapt to the changes. *The parameter k_2 is a better choice since any changes in the plant ultimately result in increase or decrease in the queue length.* Hence, k_2 can be varied to adjust the rates accordingly. The following reasoning is used in varying k_2 value.

Tuning k_2 : Examine the queue lengths in the previous 'l' number of sample times. If the queue length is monotonically increasing or decreasing, this means that the incoming rate is more than or less than the available capacity respectively. In other words, the feedback given by the controller is not changing fast enough to keep up with changes in the capacity. Thus, the value of proportionality constant k_2 should be increased. On the other hand, if queue length is oscillating around some value this means that this controller wants to reach a stable point in between. Hence, the value of k_2 should be decreased. If the queue length is neither oscillating nor monotonically increasing or decreasing then the value of k_2 will not be changed.

Chapter 4

EXPERIMENTS AND DISCUSSION OF THE RESULTS

4.1 Introduction

This chapter presents complete details of the experiments conducted and the results obtained. In Section 4.2, the mathematical model of the plant is described. In Section 4.3, a detailed description of the experimental setup and the conducted experiments is given. In Section 4.4, results obtained for H_2, H_∞, mixed H_2/H_∞ and adaptive controllers are separately presented.

4.2 Mathematical Model

Consider a bottleneck node with N source connections as shown Figure 4.1.

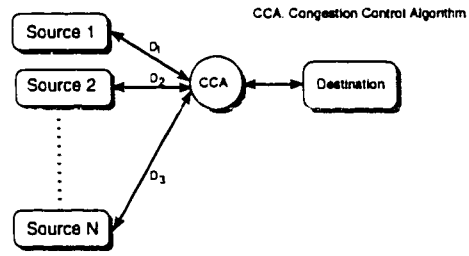


Figure 4.1: Network with N sources and one switch

Let $Q(t)$ denotes the queue length at the bottleneck node and $I_i(t)$ be the flow rate at the i^{th} source. We are assuming that the sources are greedy, that is, the sources always have some data to send and they send the data at maximum rate they are allowed to send. Later, we will relax greediness assumption by adding a Gaussian noise to the inputs.

The capacity ($C(t)$) is the rate at which data is sent out from the node. Note that the capacity at the node is the flow rate assigned by the following node in the network. The capacity is a possible input to the controller. The feedback controller assigns the flow rates ($I_i(t)\ \forall\ i = 1 \cdots N$) to the sources based on the measured queue length and the available capacity. The flow model queue length can be modeled as,

$$\dot{Q}(t) = \sum_{i=1}^{N} I_i(t - d_i) - C(t) \tag{4.1}$$

where d_i is the return trip time delay between the i^{th} source and the bottleneck node. Equation (4.1) can be discretized as

$$Q(n) - Q(n-1) = T_s(I_1(n-d_1) + ... + I_N(n-d_N) - C(n)) \qquad (4.2)$$

where T_s is the time at which feedback is updated at the sources. Equation 4.2 is the fluid approximation of the queue length. So a noise term (Q_n) should be added to the model. This term can be modelled as an additive white noise as in [24]. After some manipulations, the queue length in frequency domain is given by

$$Q(Z) = \frac{\lambda T_s}{1-\lambda}\lambda^{d_1} I_1 + \cdots + \frac{\lambda T_s}{1-\lambda}\lambda^{d_N} I_N - \frac{\lambda T_s}{1-\lambda} C \qquad (4.3)$$

where λ is equal to Z^{-1}.

The controller should also achieve *fairness* in the steady state. That is, the difference between the rates allocated to the different sources should be approximately zero. Therefore to ensure the fairness, additional outputs $(I_m - I_j)\ \forall\ j = 1 \cdots N$, $j \neq m$ are added. Where I_m is the rate allocated to the nearest source.

In the frequency domain the feedback system can be represented by transfer functions as shown in Figures 4.2. $G(Z)$ is the underlying plant and the $K(Z)$ is the controller. Clearly the plant is a MIMO system whose transfer function is given by the following equation.

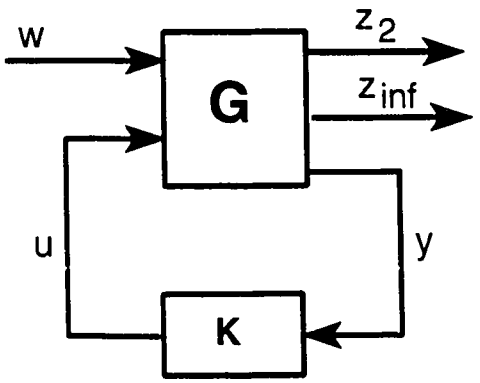


Figure 4.2: A General Feedback System

$$G(Z) = \begin{bmatrix} \frac{-\lambda T_s}{1-\lambda} & \frac{\lambda T_s}{1-\lambda}\lambda^{d_1} & \cdots & \frac{\lambda T_s}{1-\lambda}\lambda^{d_m} & \cdots & \frac{\lambda T_s}{1-\lambda}\lambda^{d_N} \\ 0 & -1 & \cdots & 1 & \cdots & 0 \\ & & \vdots & & & \\ 0 & 0 & \cdots & 1 & \cdots & -1 \\ 1 & 0 & \cdots & 0 & \cdots & 0 \\ \frac{-\lambda T_s}{1-\lambda} & \frac{\lambda T_s}{1-\lambda}\lambda^{d_1} & \cdots & \frac{\lambda T_s}{1-\lambda}\lambda^{d_m} & \cdots & \frac{\lambda T_s}{1-\lambda}\lambda^{d_N} \end{bmatrix} \tag{4.4}$$

Similarly w, y, z and u are given by following equations.

$$w = C \tag{4.5}$$

$$y = \begin{pmatrix} C \\ Q \end{pmatrix} \tag{4.6}$$

$$z_2 = z_{inf} = \begin{pmatrix} Q \\ I_m - I_1 \\ I_m - I_2 \\ \vdots \\ I_m - I_N \end{pmatrix} \tag{4.7}$$

$$u = \begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_N \end{pmatrix} \tag{4.8}$$

4.3 Experimental Setup

In this section, the experimental setup used for designing and simulating all the feedback controllers is discussed. To ease the design and the simulation of the controllers we have chosen $N = 3$. That is, three sources are connected to the node (switch). Note that for any N, the design and the simulation steps are the same. To make the problem more realistic it is assumed that these sources are at different distances from the switch. In our design we have chosen time delays as $d_2 = 2d_1$ and $d_3 = 3d_1$.

Going beyond that, the designed controller's behavior in the real time environment is observed by simulating the network as shown in the Figure 4.3. In this network

each node has its own local controller and these local controllers control all the sources connected to that node. The following five experiments are conducted on the network to test the behavior of the designed controllers in the different real time situations.

1. **Experiment 1 :** In this experiment the network is simulated with the greediness assumption. That is, all the sources send exactly at the same rate (I_n) requested by the controller. This is an ideal case.

2. **Experiment 2 :** In this experiment the greediness assumption is eliminated. So this experiment is more realistic. Usually, sources may not be able to send at the requested rate. Even if the sources send at the requested rate, there is a certain (even though very small) probability of cell loss and cell delay. Thus, the data arrives at the node at a slightly different rate than the requested rate. However, it should be noted that these deviations are small. This experiment is simulated by adding a Gaussian noise to the flow rates.

3. **Experiment 3 :** In this experiment the network is simulated for the situation when the communication between one or more nodes is lost. In real life, this situation is quite possible. The reason can be anything, for instance, the communication link might have broken or the software responsible for communication might have crashed, etc. Thus, this experiment simulates the situation when the communication channel between node 2 and node 4 is

broken at the 50^{th} sampling time and the communication channel between node 3 and node 4 is broken at the 55^{th} sampling time. This experiment can also be interpreted as the case when some of the connected sources suddenly terminates their connection.

4. **Experiment 4 :** In this experiment the network is simulated when the feedback sent to the sources is lost. This is experiment is also realistic since there is a certain probability of losing the RM cells in which feedback is sent from the nodes to the sources. Hence, in this experiment it is assumed that feedback sent from node 4 to node 1 was lost during the 12^{th} to the 45^{th} sampling time and feedback sent from node 4 to node 2 was lost during the 22^{th} to the 55^{th} sampling time. Thus, node 1 and node 2 continues sending data at the rate just before the 12^{th} and the 22^{th} sampling time respectively. But, at the 45^{th} sampling time node 2 and at the 55^{th} sampling time node 3, receives the feedback again from the node 4 and readjusts their data rates.

5. **Experiment 5 :** In all the above experiments each practical event is individually simulated. But in practice, all the above events can happen at any time. Therefore, in this experiment all the following events are simulated simultaneously.

   (a) The sources send data at slightly different rate rather than at the requested rate.

(b) The feedback sentfrom node 4 to node 2 is lost during the 10^{th} to the 50^{th} sampling times.

(c) The communication channel between node 3 and node 4 is broken at the 45^{th} sampling time.

The staircase signal shown in Figure 4.4 is the external controller input signal (available capacity) used to test the designed controllers in all the simulations. The reason behind selecting this kind of input is that the true behavior of the controller becomes evident when the input changes abruptly. Experiment 5 is also conducted with the random input signal shown in Figure 4.5. This input is generated by mixing sinusoidal, step and saw tooth inputs. This type of input is chosen merely to ensure that the designed controllers work properly in the real time environment.

4.4 Controller Design and Results

The mathematical model of our plant is in discrete time. However, control design tools and algorithms are well developed for continuous time systems. So, we have transformed our discrete time plant to a continuous time plant by using bi-linear transformation and designed controllers for the continuous time plant using the LMI tool box of MATLAB. This transformation doesn't effect the H_∞ norm since the H_∞ norm is the same for both continuous and discrete systems. However, the continuous time H_2 norm has to be multiplied with $\frac{\sqrt{2}}{s+1}$ to actually minimize discrete

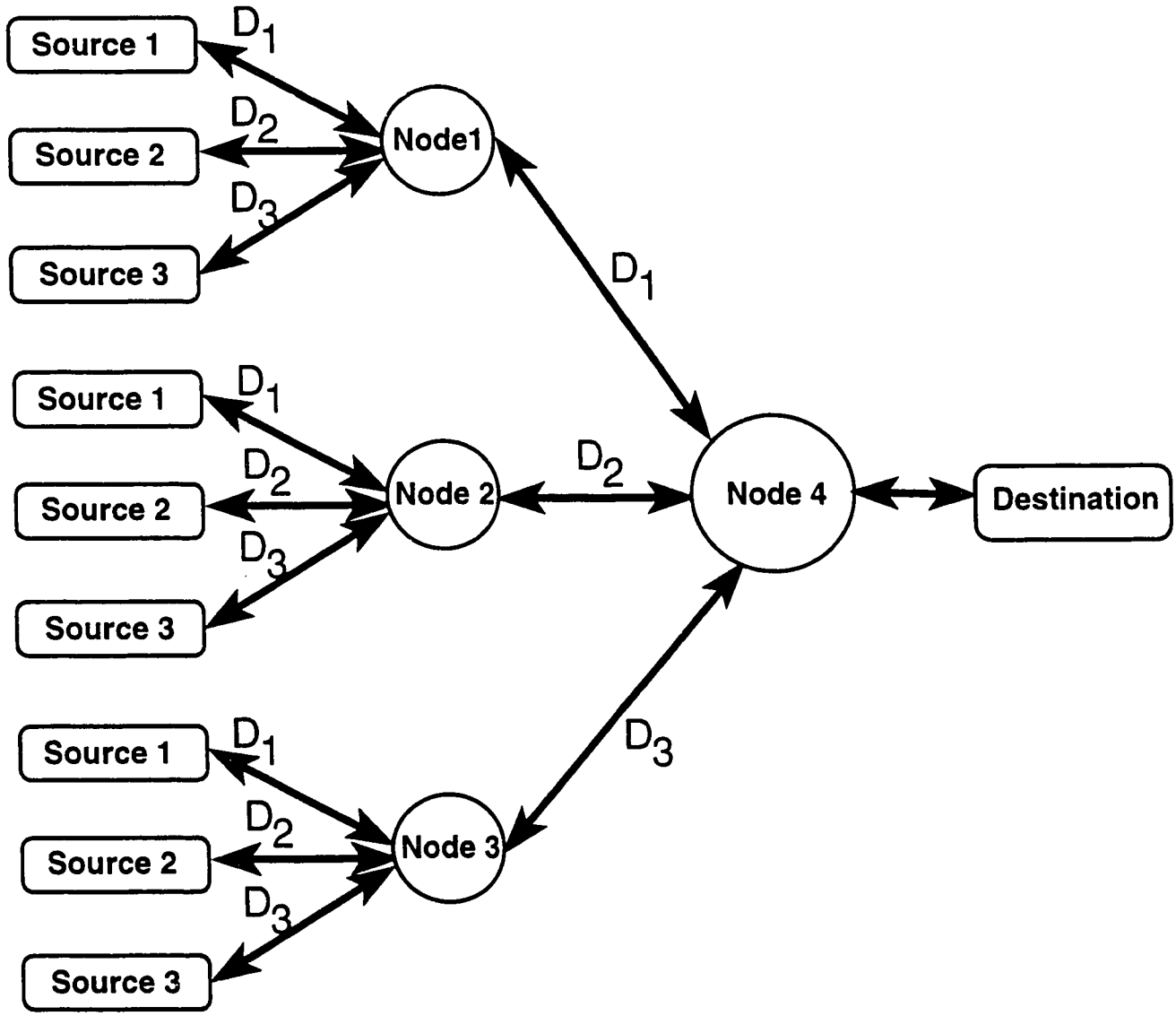


Figure 4.3: Network used in all experiments

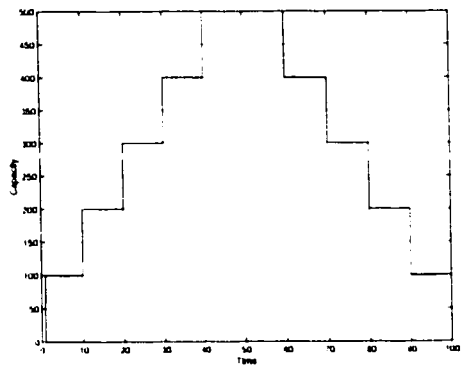


Figure 4.4: External input signal used for testing the controllers

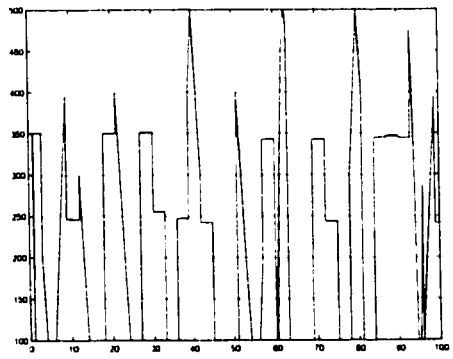


Figure 4.5: Random input signal used in Experiment 5

time H_2 norm. Similarly, in mixed H_2/H_∞ problem the outputs where the H_2 norm should be minimized should also be multiplied by $\frac{\sqrt{2}}{s+1}$. The designed continuous time controllers are transformed back to discrete time and all the simulations are conducted on the original discrete time plant.

4.4.1 H_2 Controller

A H_2 controller is designed to minimize the H_2 norm of the transfer function from the capacity ($C(t)$) to the queue length ($Q(t)$) and to the difference of the input flow rates ($I_1 - I_2$, $I_1 - I_3$). The single node network shown in Fig. 4.1 is simulated with the designed H_2 controller using the signal shown in Figure 4.4 as input. The obtained simulation results are displayed in Figure 4.6. From Figure 4.6(a) we can infer that the queue length increases by 1.8 units when the capacity decreases by 100 units (or at the rate of -0.018 with capacity). Figure 4.6(b) shows the flow rates assigned by the controller. Lets analyze the flow rates carefully. Let C be the input to the controller, that is, the available capacity is C. The controller should efficiently use the available bandwidth (C) without sacrificing the fairness. Ideally the controller's feedback to the sources should be $I_1 = I_2 = I_3 = \frac{C}{3}$. But, as the sources are at different distances from the controller, there will be some time delay before the sources adjust their rates according to the feedback received from the controller. This causes the differences between the input flow rates. But, the controller tries to equalize the flow rates. The values obtained in the steady state for

$C = 500$ are $I_1 = 166.7$, $I_2 = 166.68$and $I_3 = 166.62$. These flow rates are almost equal. So, we can say that the H_2 controller has achieved the first design objective (fairness). The utilization of the network is given by

$$\rho = \min\left(\frac{\sum_{i=1}^{3}\sum_{j=1}^{3} I_{ij}}{C},\quad 1\right). \tag{4.9}$$

In this case utilization (ρ) is equal to one. This means, there is hundred percent utilization of the capacity(C) in the steady state and this is achieved with minimum changes in the queue length. Hence, we can conclude that the H_2 controller is satisfying all the design objectives. Figures 4.6(c) and 4.6(d) show the difference between the assigned flow rates, which are plotted to show how fast H_2 controller achieves fairness. These plots will be compared later with the other controller's results.

The designed H_2 controller's performance further needs to be investigated before commenting on the controller's behavior in the real time networked environment. So the designed H_2 controller is tested by conducting the previously described five experiments.

1. **Experiment 1 :** The results of this experiment with the designed H_2 controller are shown in Figure 4.7. The first row in this figure corresponds to the outputs (flow rates assigned by the controller and queue length [1]) of node 1.

[1]Note that all the plots and tabulated values of queue length in this thesis are the deviations of the queue length from the desired constant value. Hence they can be negative.

Similarly, the second row, the third row, and the fourth row correspond to the outputs of node 2, node 3, and node 4 respectively.

Let us analyze these simulation results further. Again let us assume the available capacity at the main node (node 4) is C. The local controller at this node sends the feedback (I_1, $I_2$and I_3) to the three different sources based on the available capacity (C). Now these I_1, I_2 and I_3 are the maximum available capacities to the node 1, node 2 and node 3 respectively. Subsequently, the local H_2 controllers at these nodes divide these flow rates among their sources. The values of the flow rates assigned by the local controllers for $C = 500$ are shown in Table 4.1 and the corresponding queue lengths are shown in Table 4.2, where I_{ij} is the flow rate assigned by the local controller at node i to source j. From this table, it is obvious that the designed H_2 controller is working very efficiently.

2. **Experiment 2 :** This experiment is conducted with the designed H_2 controller and the corresponding simulation results are shown in Figure 4.8. These results are similar to the results of the previous experiment. The sole difference is, small perturbations in the queue lengths and in the assigned flow rates arose because of the uncertainties in the flow rates from the sources. Of course, this

Table 4.1: Feedback given by the local H_2 controller in Exp. 1 at $T_s = 50$

F/W Rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	166.7	166.68	166.62	55.61	55.56	55.52	55.552	55.545	55.525	55.576	55.57	55.557	1.0

Table 4.2: Deviations of queue length from the desired value at the different nodes in Exp. 1 at $T_s = 50$

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	−5.55	−2.967	−2.355	−8.90

is natural and cannot be eliminated without sacrificing the utilization.

3. **Experiment 3 :** The results of this experiment with the designed H_2 controller are shown in Figure 4.9. From this figure we can see that at $T_s = 50$, the channel between node 2 and node 4 is broken. Now the local H_2 controller at node 4 has to divide the available capacity between node 1 and node 3. From Table 4.3 the obtained values of I_1 and I_3 are 250.04 and 249.98 respectively. Thus, the controller is satisfying the efficiency constraint. Note that the controller feedback to the node 2 (I_2) is still 250.02. This is essential for the controller to maintain the fairness among the sources. Nevertheless, this feedback (I_2) will never reach to node 2. Another point is that queue length at node 2 has suddenly dropped to zero since the local controller immediately requests its sources to stop sending data. At $T_s = 55$, the channel between node 3 and node 4 is also broken. Hence, the H_2 controller allocates all the capacity to node 1. But, to maintain fairness the controller approximates I_2 and I_3 to I_1. The complete related values are given in Tables 4.3 and 4.4.

Table 4.3: Feedback given by the local H_2 controllers in Exp. 3 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	250.04	250.02	249.98	83.4	83.35	83.1	0	0	0	83.34	83.32	83.3	1.0
$T_s = 55$	500.02	500	449.98	166.8	166.7	166.5	0	0	0	0	0	0	1.0

Table 4.4: Deviations of queue length from the desired value at the different nodes in Exp. 3 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	−8.33	0	−4.45	−17.2
$T_s = 55$	−16.67	0	0	−42.19

4. **Experiment 4 :** The experimental results obtained for this case with the designed H_2 controller are displayed in Figure 4.10. Recall that the feedback sent to node 1 is lost during the 12^{th} to 45^{th} sampling time and the feedback sent to node 2 is lost during the 22^{th} to 55^{th} sampling times. Hence, node 1 and node 2 continue to send at the rate prior to the 12^{th} and the 22^{th} sampling time respectively. During these periods the unused capacity should be distributed among the other sources. To clarify the situation, let us examine the rates at $T_s = 22$. At this moment the feedback to both node 1 and node 2 is lost. Hence, node 1 and node 2 continue to send at the rates I_1 ($I_{11} + I_{12} + I_{13} = 66.67$) and I_2 ($I_{21} + I_{22} + I_{23} = 116.65$). So, the remaining capacity ($C - I_1 - I_2 = 116.58$) should be allocated to node 1. From Table 4.5, the value of I_3 can be read as 116.67 which is approximately same as the calculated one. Other events can be interpreted in the same way.

5. **Experiment 5 :** This experiment is conducted with the designed H_2 controller

Table 4.5: Feedback given by the local H_2 controller in Exp. 4 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	66.68	66.67	66.64	22.24	22.22	22.21	22.24	22.22	22.21	22.23	22.23	22.22	1.0
$T_s = 22$	116.68	116.68	116.67	22.24	22.24	22.21	38.89	38.89	38.87	38.9	38.9	38.88	1.0
$T_s = 45$	191.68	191.67	191.64	63.95	63.9	63.85	38.89	38.89	38.87	63.9	63.9	63.88	1.0
$T_s = 55$	166.67	166.68	166.62	55.61	55.55	55.52	55.55	55.55	55.52	55.58	55.57	55.55	1.0

Table 4.6: Deviations of queue length from the desired value at the different nodes in Exp. 4 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−2	−1.187	−1.187	−3.561
$T_s = 22$	−2	−2.077	−2.078	−7.007
$T_s = 45$	−6.4	−2.077	−3.413	−11.4
$T_s = 55$	−5.5	−2.968	−2.968	−8.903

and the corresponding simulation results are shown in Figure 4.11. In this experiment, as explained before, all the fore-mentioned events are simulated simultaneously. This experiment is also conducted with the random input shown in Figure 4.5 and the corresponding results are shown in Figure 4.12. All these results can be interpreted in a similar way as explained in the previous experiments. From Figures 4.11 and 4.12, it can be inferred that the H_2 controller is perfectly handling the situations without sacrificing fairness and efficiency.

Table 4.7: Feedback given by the local H_2 controller in Exp. 5 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	62.55	60.7	64.05	22.11	21.34	22.07	22.23	22.92	22.20	22.50	22.37	25.91	1.0
$T_s = 45$	431.31	432.38	430.35	143.65	142.85	143.19	22.23	22.92	25.2	0	0	0	1.0
$T_s = 55$	250.2	251.2	249.8	83.66	83.76	84.03	82.9	87.01	83.98	0	0	0	1.0

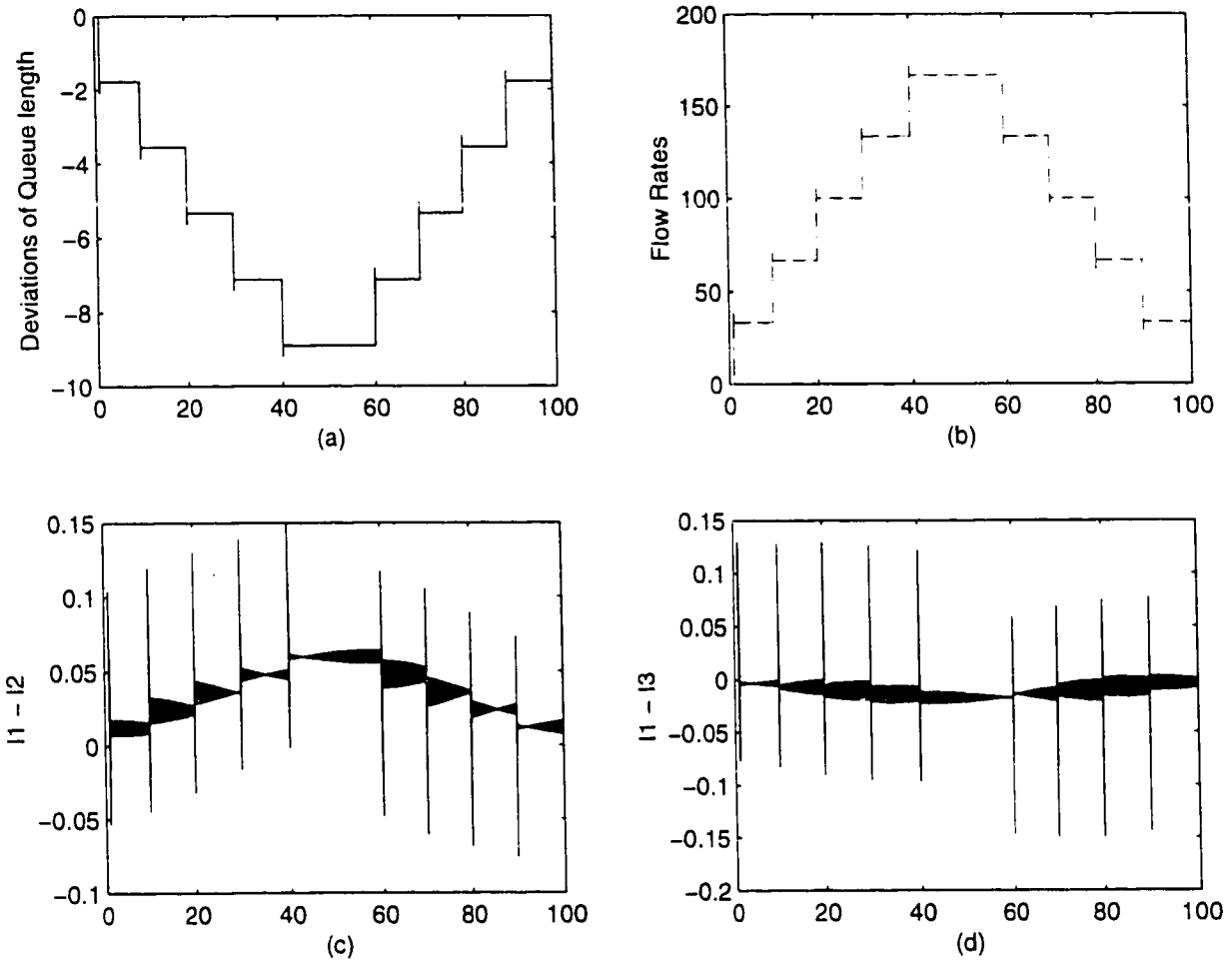


Figure 4.6: The Simulation Results of H_2 Controller

Table 4.8: Deviations of queue length from the desired value at the different nodes in Exp. 5 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−2.16	−1.22	−1.56	−3.31
$T_s = 45$	−14.21	−1.21	0	−35.17
$T_s = 55$	−8.34	−4.52	0	−17.19

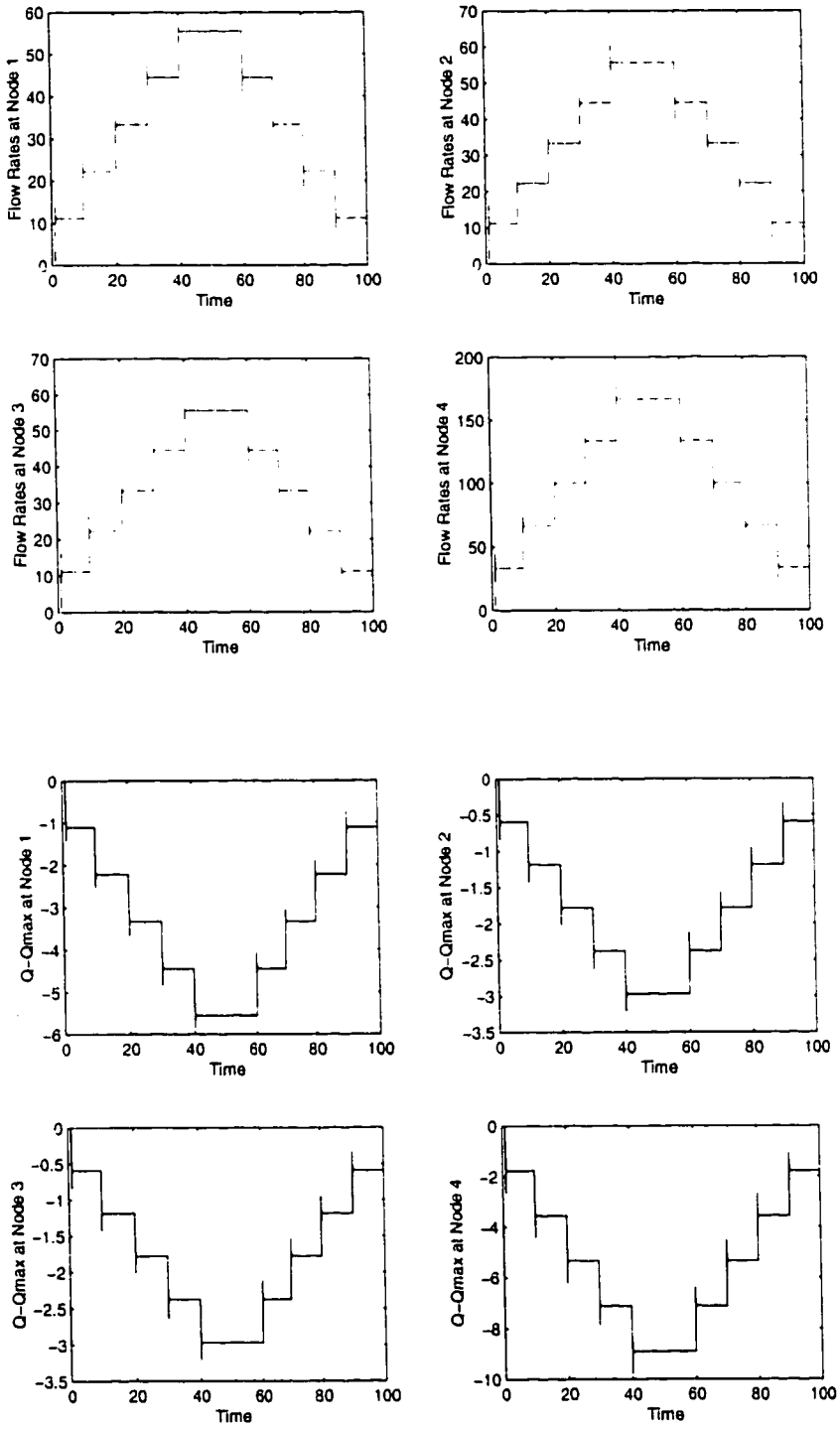


Figure 4.7: The H_2 Simulation results of Exp. 1

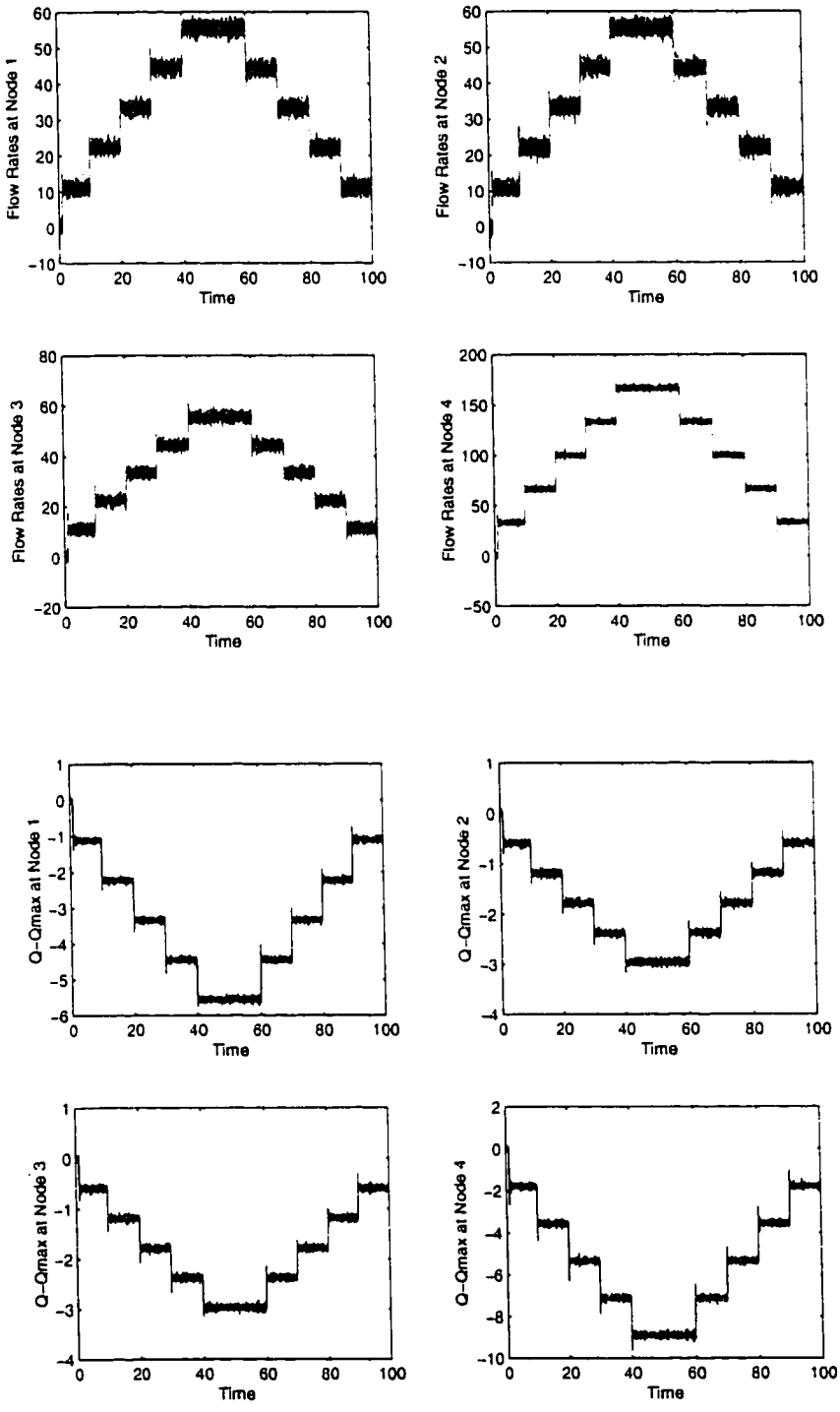


Figure 4.8: The H_2 Simulation results of Exp. 2

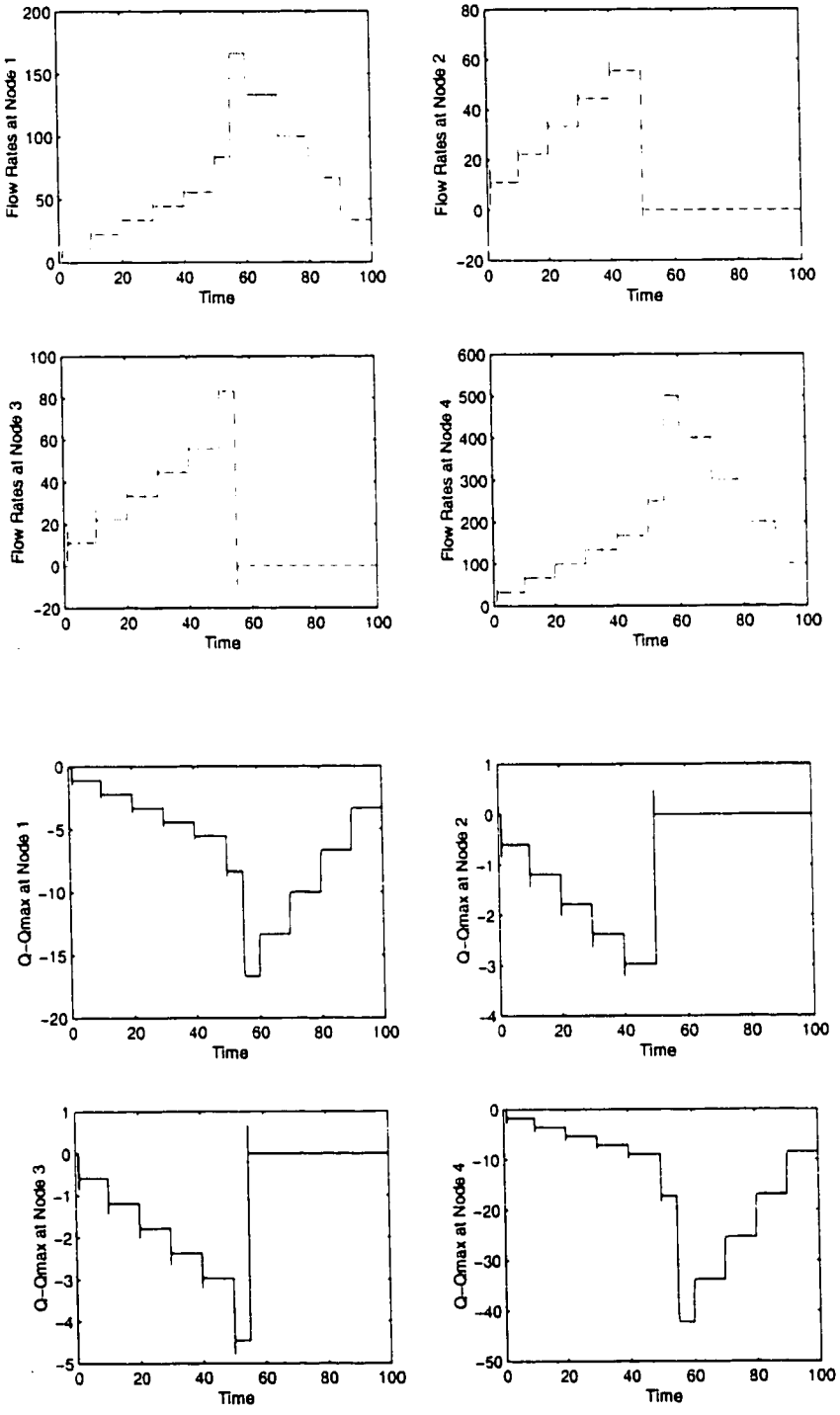


Figure 4.9: The H_2 Simulation results of Exp. 3

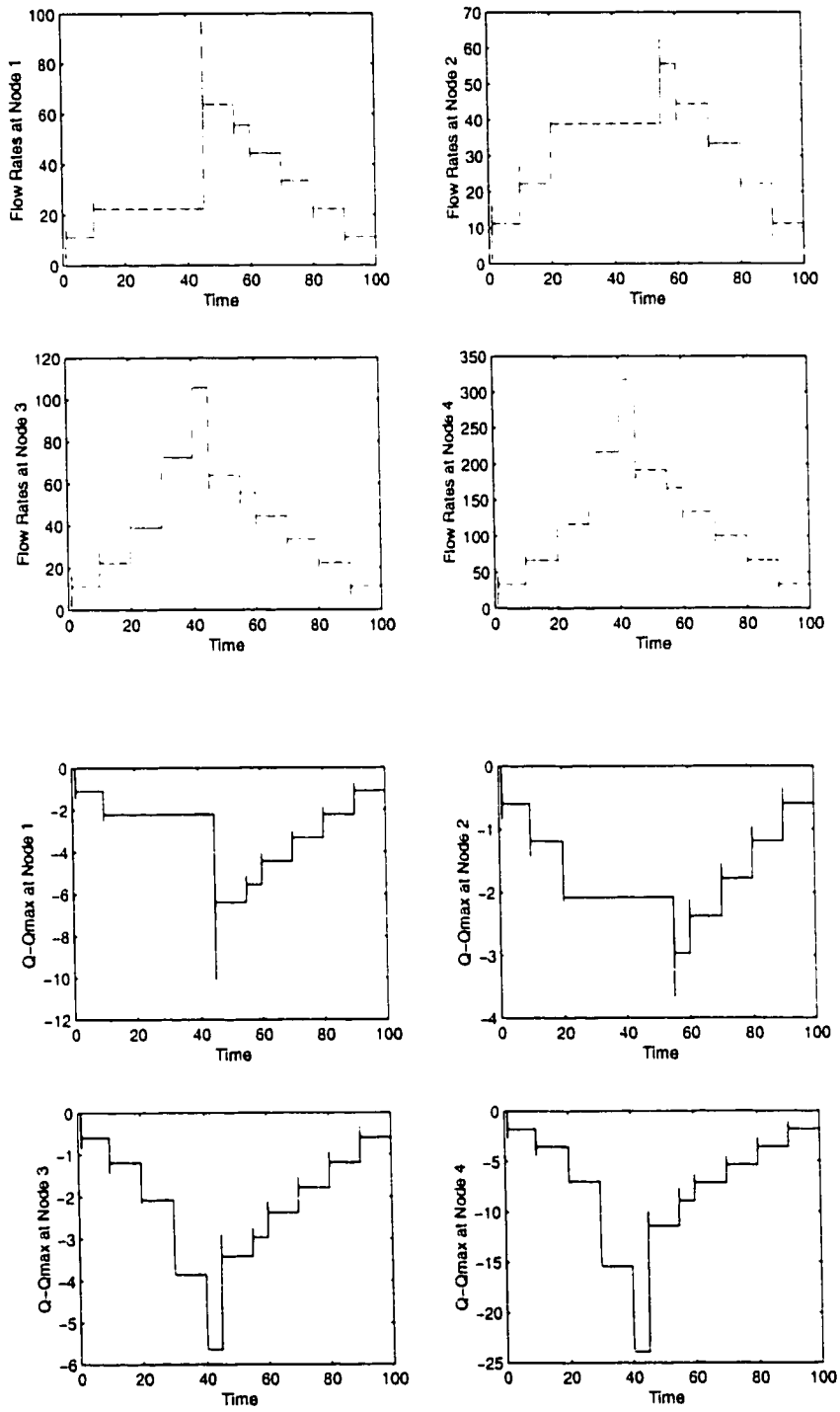


Figure 4.10: The H_2 Simulation results of Exp. 4

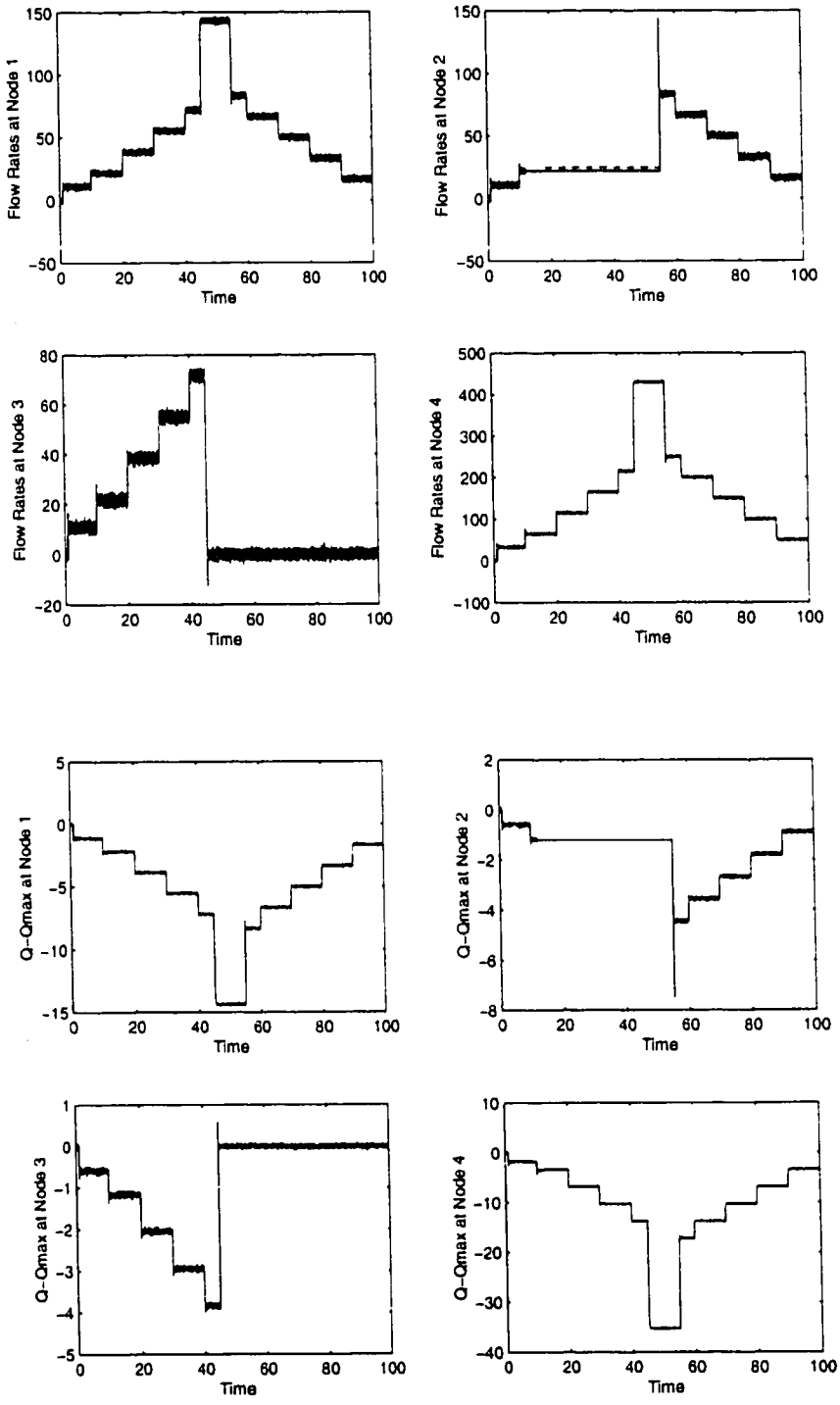


Figure 4.11: The H_2 Simulation results of Exp. 5

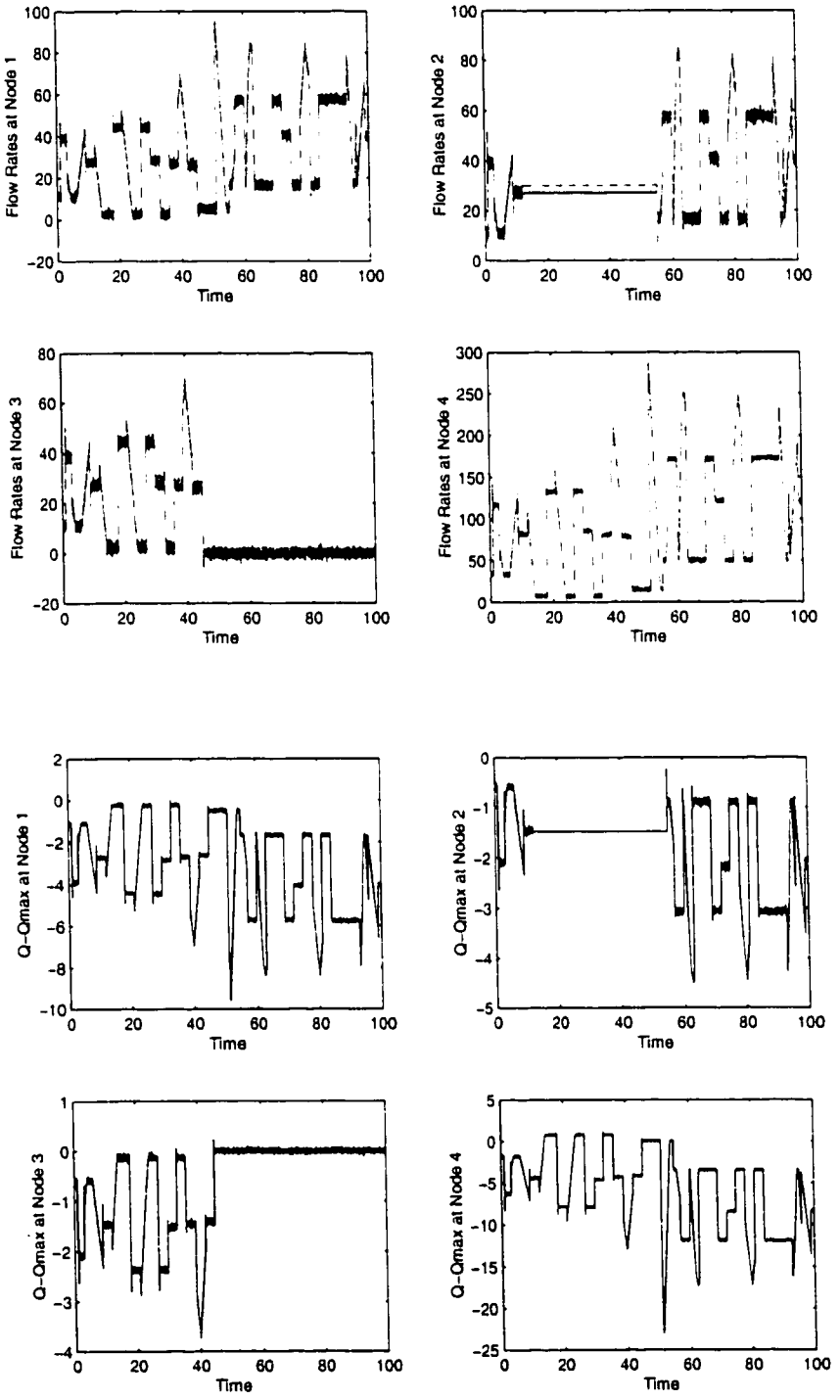


Figure 4.12: The H_2 Simulation results of Exp. 5 with the random input

4.4.2 H_∞ Controller

A H_∞ sub-optimal controller is designed to minimize the H_∞ norm of the transfer function from the capacity ($C(t)$) to the queue length ($Q(t)$) and to the difference of the input flow rates ($I_1 - I_2$, $I_1 - I_3$). The designed controller is simulated with the same input signal shown in Figure 4.4. The obtained simulation results are shown in Figure 4.13. From Figure 4.13(a), we can deduce that queue length changes at the rate of -0.022 with capacity. Figure 4.13(b) shows the flow rates assigned by the H_∞ controller. The flow rates obtained with $C = 500$ are $I_1 = 166.7$, $I_2 = 166.68$and $I_3 = 166.62$. These flow rates are almost equal. This means the first design objective of the H_∞ controller has been accomplished.

The utilization of the network can be found using the Eq. (4.9). Even in this case the utilization is one. That is, the H_∞ controller is utilizing all the available capacity (C) in the steady state. The primary difference between the results of H_2 and H_∞ controllers is the overshoots and undershoots in the flow rates when the capacity changes suddenly. These overshoots and undershoots are small in the H_∞ case. Another difference can be visualized by comparing Figures 4.13(c) and 4.13(d) with the corresponding H_2 results in Figure 4.6. Again these plots are much smoother than the H_2 plots. Remember that this characteristic is called the responsiveness of the controller and it is a desired feature. The point which is in favor of the H_2 controller is that it needs small buffers at the nodes than that of the H_∞ controller.

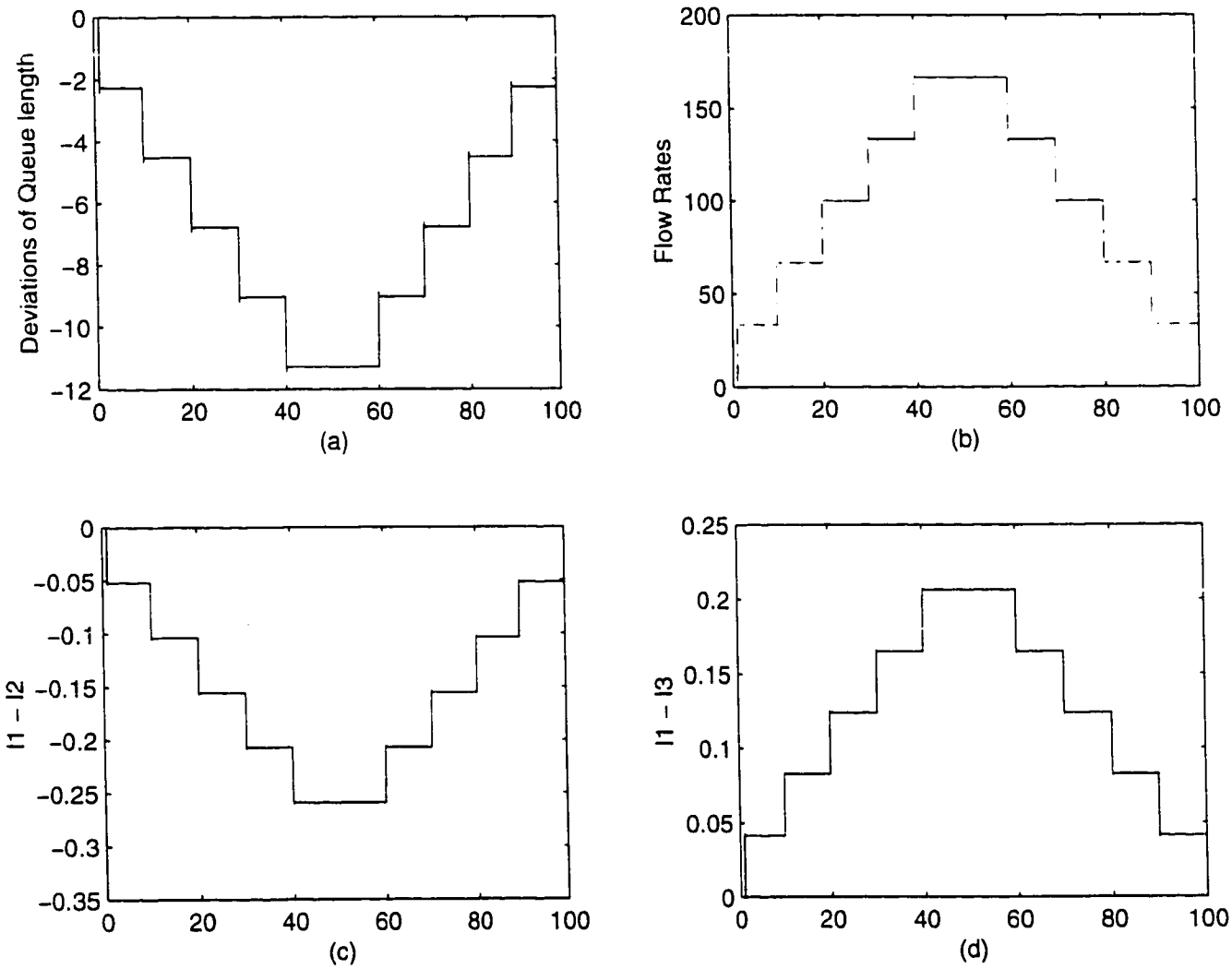


Figure 4.13: The Simulation results of H_∞ controller

Table 4.9: Feedback given by the local H_∞ controllers in Exp. 1 at $T_s = 50$

F/W Rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	166.65	166.90	166.44	55.54	55.62	55.48	55.63	55.71	55.56	55.47	55.56	55.40	1.0

Table 4.10: Deviations of queue length from the desired value at the different nodes in Exp. 1 at $T_s = 50$

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	-8.32	-3.77	-3.76	-11.29

The same five experiments are conducted with the designed the H_∞ controller. The obtained simulation results are similar to the H_2 controller's results except that the responsiveness of the H_∞ controller is better and the queue length needed at the nodes is slightly higher.

1. **Experiment 1 :** The results of this experiment with the designed H_∞ controller are shown in Figure 4.14. The local H_∞ controller at main node sends the feedback (I_1, $I_2$and I_3) to the three different sources based on the available capacity (C). Now feedbacks I_1, I_2 and I_3 are the maximum available capacities to node 1, node 2 and node 3 respectively. Subsequently, the local controllers at these nodes divide these flow rates among their sources. The values of the flow rates assigned by the local H_∞ controllers for $C = 500$ (at $T_s = 50$) are shown in Table 4.9 and the corresponding queue lengths are shown in Table 4.10.

Table 4.11: Feedback given by the local H_∞ controller in Exp. 3 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	250.15	250.53	249.86	83.37	83.49	83.28	0	0	0	83.28	83.49	83.28	1.0
$T_s = 55$	500	500.8	499.5	166.6	166.9	166.5	0	0	0	0	0	0	1.0

2. **Experiment 2 :** This experiment is conducted with the designed H_∞ controller and corresponding simulation results are shown in Figure 4.15. These results are similar to the results of the previous experiment. The only difference is small perturbations in the queue lengths and in the assigned flow rates.

3. **Experiment 3 :** The results of this experiment with the designed H_∞ controller are shown in Figure 4.16. Since at $T_s = 50$ the channel between node 2 and node 4 is broken, the H_∞ controller has to divide the available capacity between node 1 and node 3. The obtained values of I_1 and I_3 are 250.15 and 249.86 respectively. Hence, the controller is satisfying the efficiency constraint. As explained before in the H_2 case, the controller still assigns feedback to node 2 ($I_2 = 250.53$), which is essential for the controller to maintain the fairness among the sources. The channel between node 3 and node 4 is also broken at the 55^{th} sampling time. Thus, the H_∞ controller allocates all the capacity to node 1. The related values can be found in Tables 4.11 and 4.12.

4. **Experiment 4 :** The experimental simulation results obtained for this case with the designed H_∞ controller are displayed in Figure 4.17. Lets examine the rates at $T_s = 22$. At this moment the feedback sent to node 1 and node

Table 4.12: Deviations of queue length from the desired value at the different nodes in Exp. 3 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	−12.5	−13.39	−5.64	−23.77
$T_s = 55$	−25	0	0	−61.1

Table 4.13: Feedback given by the local H_∞ controller in Exp. 4 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	66.66	66.76	66.56	22.22	22.25	22.19	22.25	22.29	22.24	22.19	22.22	22.16	1.0
$T_s = 22$	116.5	116.83	116.50	22.22	22.25	22.19	38.94	39	38.89	38.83	38.89	38.78	1.0
$T_s = 45$	191.7	192	191.47	63.89	63.98	63.82	38.94	39	38.99	63.81	63.91	63.73	1.0
$T_s = 55$	165.15	165.41	164.91	54.11	54.19	54.06	54.87	54.95	54.80	54.73	54.81	54.66	1.0

2 is lost. Hence, node 1 and node 2 continue to send at the rates I_1 ($I_{11} + I_{12} + I_{13} = 66.66$) and I_2 ($I_{21} + I_{22} + I_{23} = 116.82$). So, the remaining capacity ($C - I_1 - I_2 = 116.52$) should be allocated to node 3. From Table 4.13, the value of I_3 can be read as 116.50 which is approximately same as the calculated one. Other events can be interpreted in the same way and all the related values are given in Tables 4.13 and 4.14.

5. **Experiment 5 :** This experiment is conducted with the designed H_∞ controller using inputs in Figures 4.4 and 4.5 and the corresponding simulation results are shown in Figures 4.18 and 4.19 respectively. From these results, it

Table 4.14: Deviations of queue length from the desired value at different nodes in Exp. 4 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−3.33	−1.51	−1.5	−4.51
$T_s = 22$	−3.3	−2.64	−2.62	−9.26
$T_s = 45$	−9.58	−2.64	−4.32	−15
$T_s = 55$	−8.10	−3.67	−3.66	−11

Table 4.15: Feedback given by the local H_∞ controller in Exp. 5 at the critical times

F/W rates	l_1	l_2	l_3	l_{11}	l_{12}	l_{13}	l_{21}	l_{22}	l_{23}	l_{31}	l_{32}	l_{33}	ρ
$T_s = 12$	66.42	66.62	66.43	22.3	22.3	22.2	22.22	22.32	22.5	22.23	22.27	22.23	1.0
$T_s = 45$	432.95	433.67	433.28	144.33	144.53	144.19	21.64	22.32	22.49	0	0	0	1.0
$T_s = 55$	249.9	249.8	249.4	83.3	83.3	83.2	83.4	83.6	83.3	0	0	0	1.0

Table 4.16: Deviations of queue length from the desired value at the different nodes in Exp. 5 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−3.32	−1.51	−1.50	−4.5
$T_s = 45$	−21.64	−1.51	0	−51.1
$T_s = 55$	−12.5	−5.56	0	−23.7

can be concluded that the designed H_∞ controller is *also* perfectly handling situation without sacrificing fairness and efficiency.

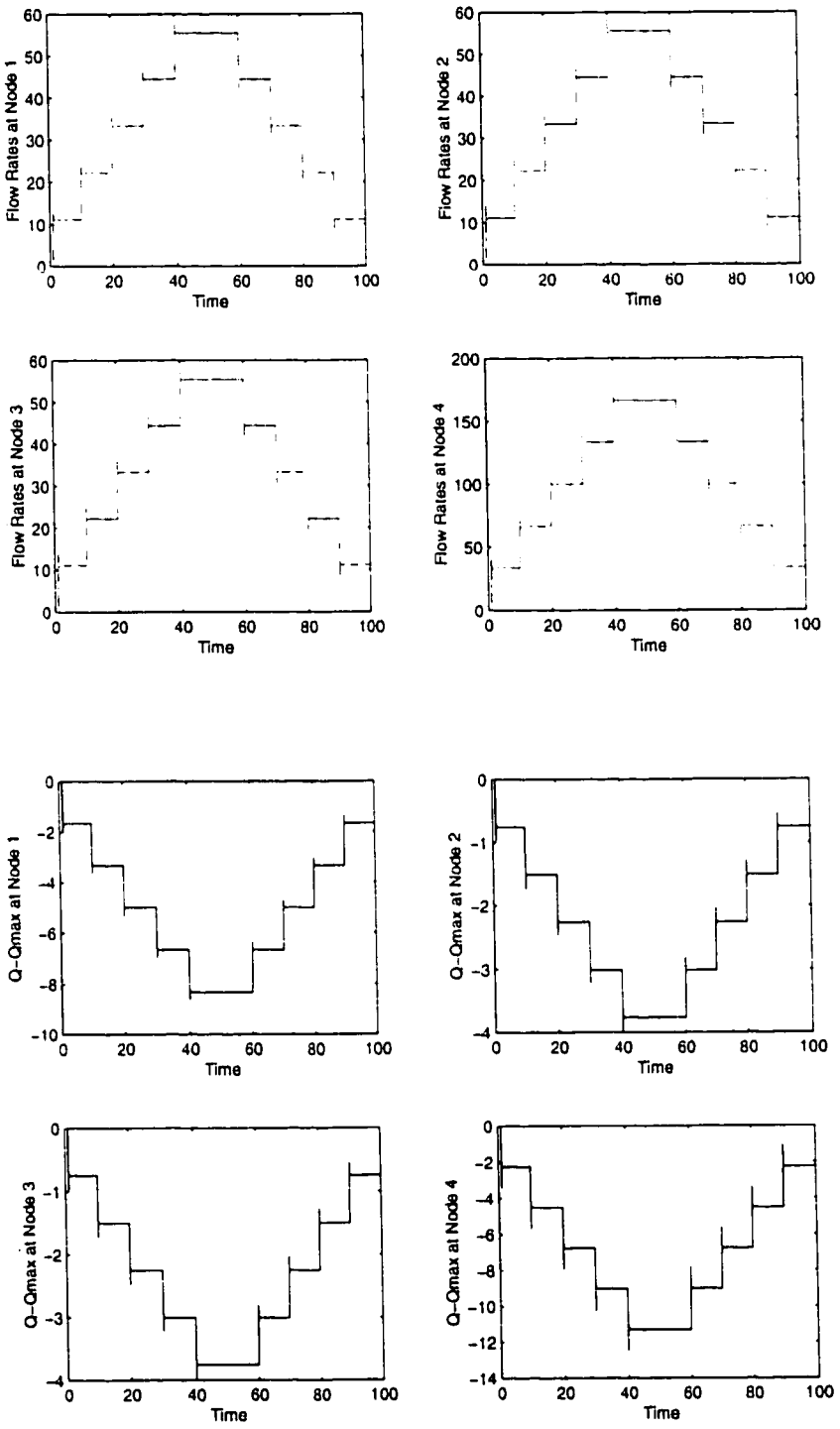


Figure 4.14: The H_∞ Simulation results of Exp. 1

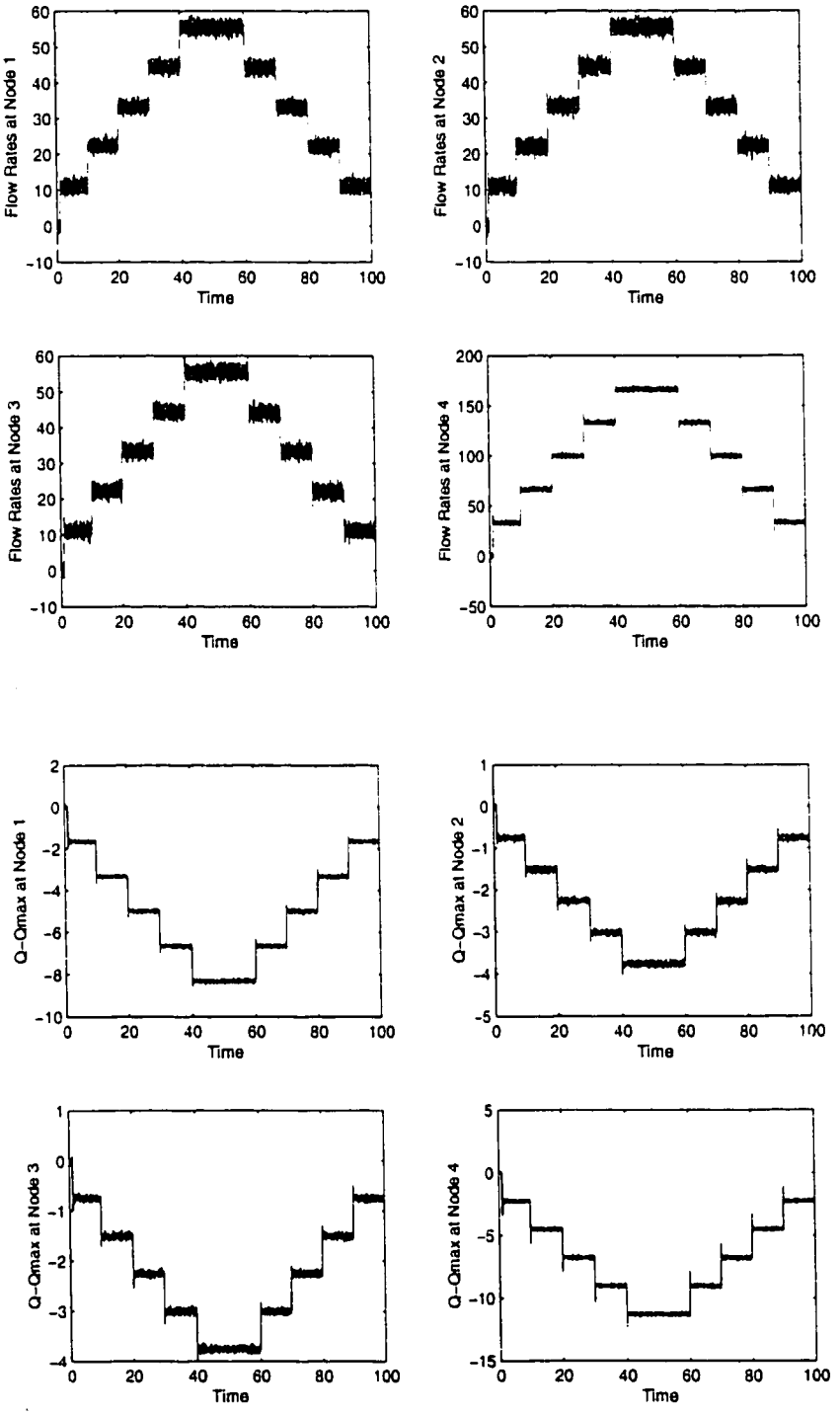


Figure 4.15: The H_∞ Simulation results of Exp. 2

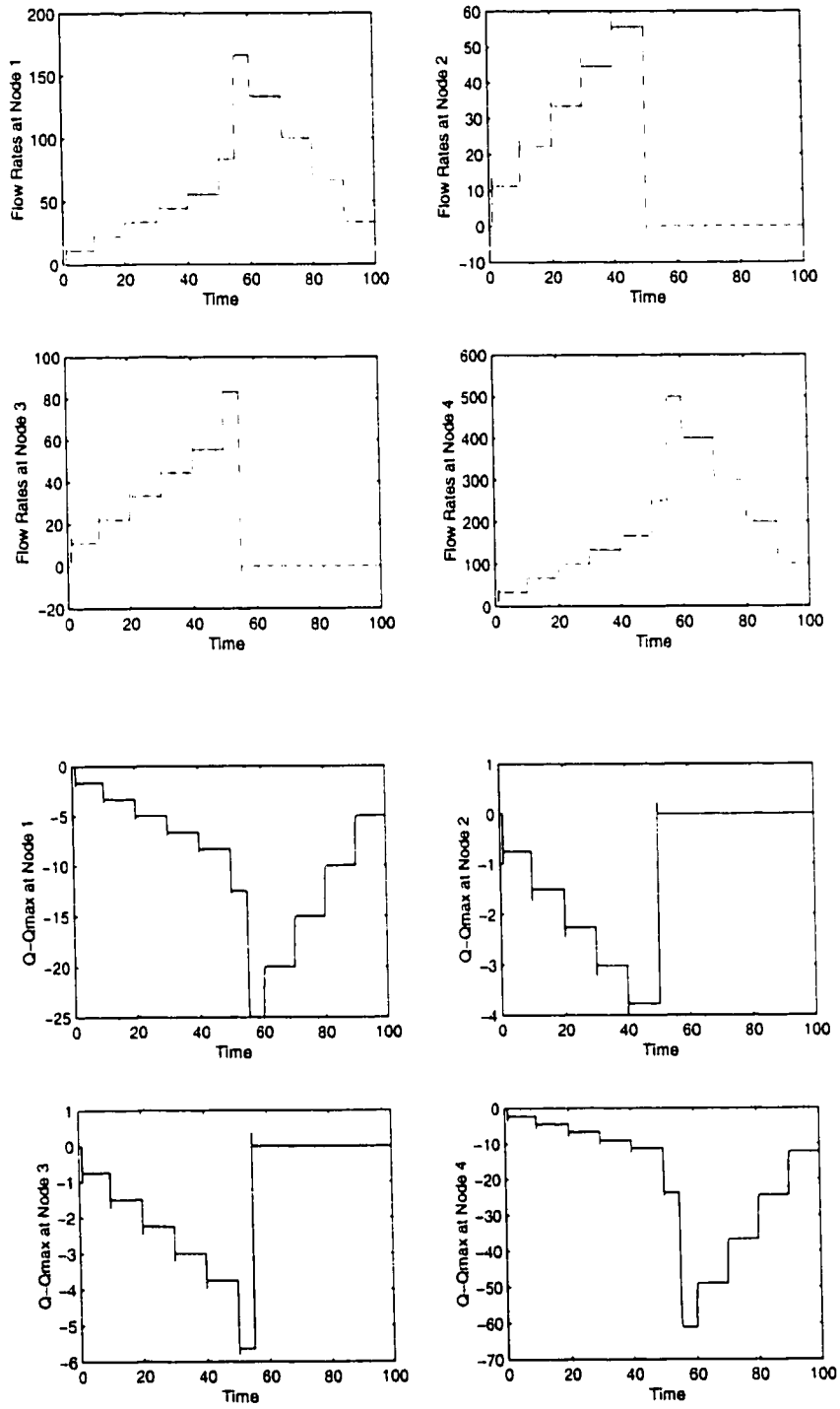


Figure 4.16: The H_∞ Simulation results of Exp. 3

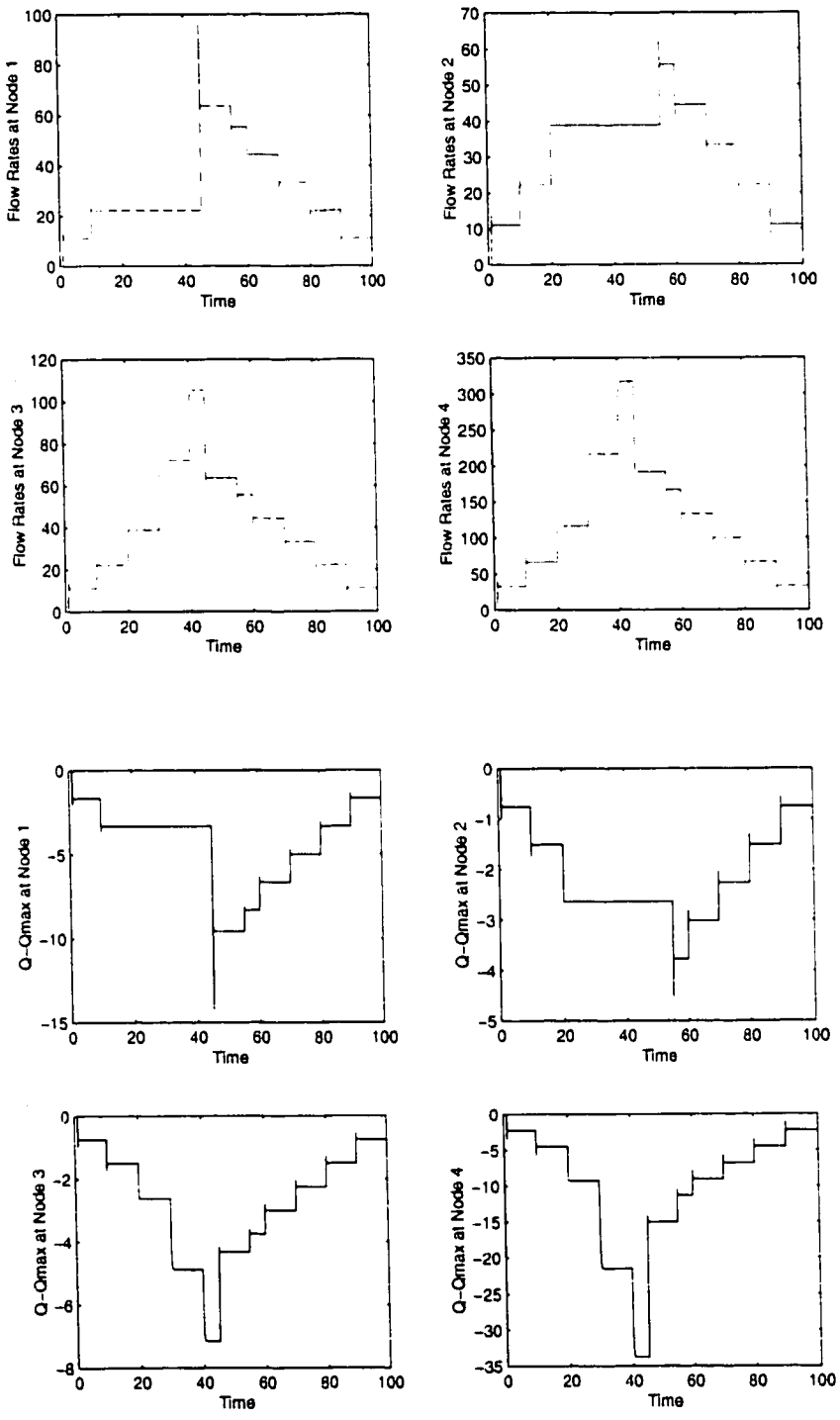


Figure 4.17: The H_∞ Simulation results of Exp. 4

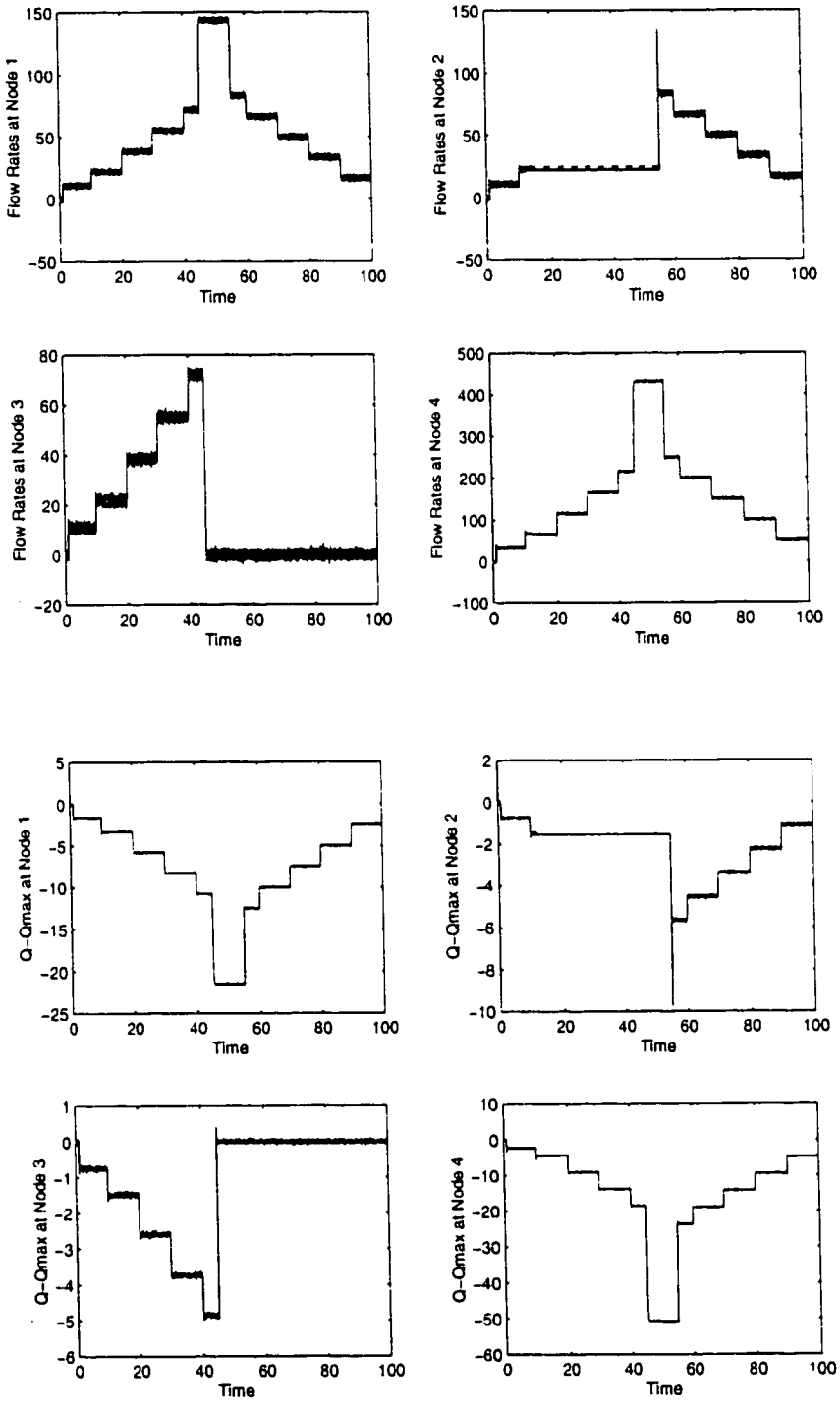


Figure 4.18: The H_∞ Simulation results of Exp. 5

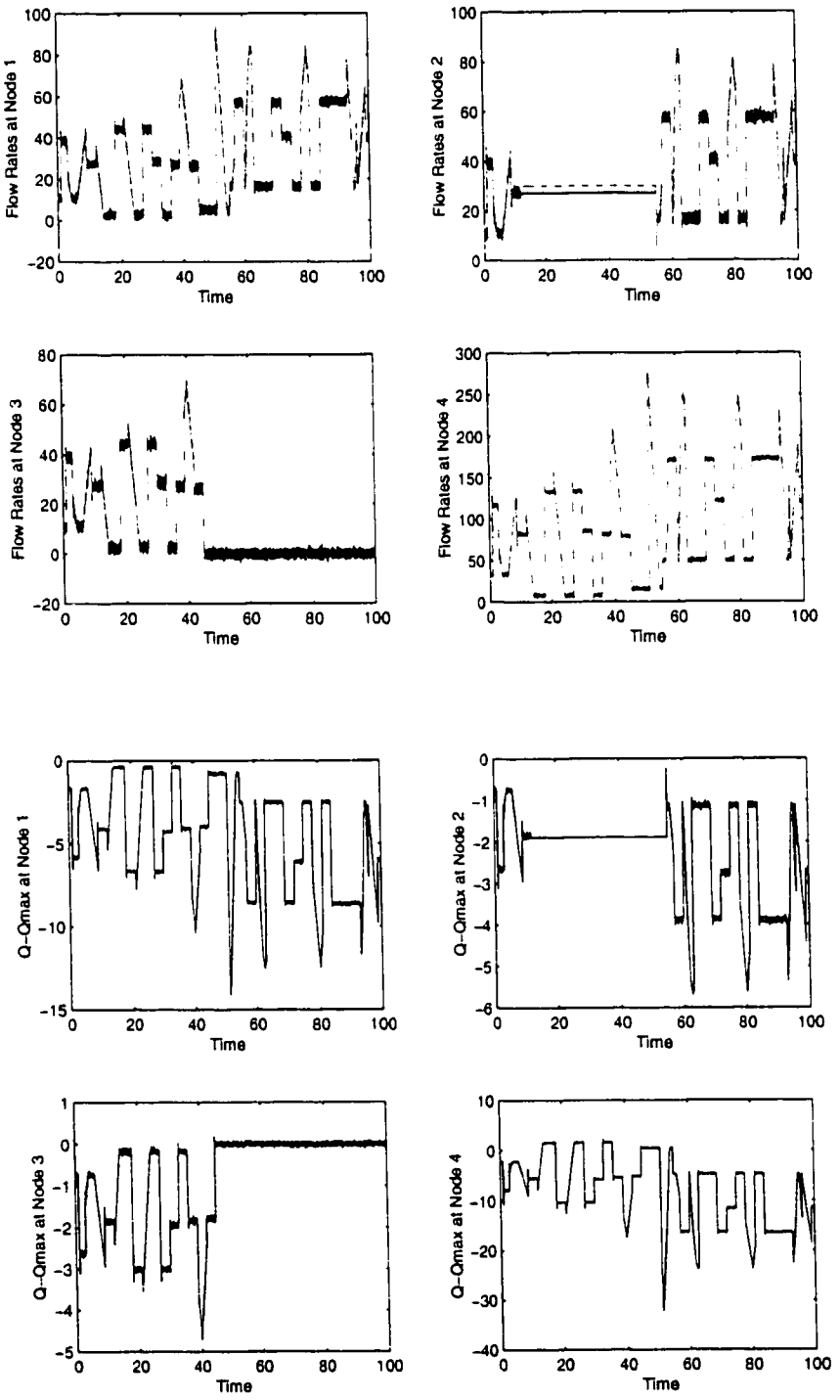


Figure 4.19: The H_∞ Simulation results of Exp. 5 with the random input

4.4.3 Mixed H_2/H_∞ Controller

A mixed H_2/H_∞ controller is designed to minimize the sum of square of the H_∞ and the H_2 norm of the transfer function from the capacity ($C(t)$) to the queue length ($Q(t)$) and from C(t) to the difference of the input flow rates ($I_1 - I_2$, $I_1 - I_3$). The simulation results are shown in Figure 4.20. From Figure 4.20(a), we can infer that queue length changes at the rate of -0.0199 with the capacity. Figure 4.13(b) shows the flow rates assigned by the mixed H_2/H_∞ controller and the flow rates obtained for $C = 500$ are $I_1 = 166.7$, $I_2 = 166.68$, and $I_3 = 166.21$. Since these flow rates are almost equal, the first design objective of the mixed H_2/H_∞ controller is accomplished.

The utilization of the network can be calculated by using the same equation (4.9). Even the mixed H_2/H_∞ controller is utilizing all the available capacity (C) in the steady state. As expected the results of the mixed H_2/H_∞ controllers have advantages of both the H_2 and H_∞ controllers. This controller needs less queue length at the nodes than that of the H_∞ controller and more queue length than that of the H_2 controller. Similarly the responsiveness (from Figures 4.20(c) and 4.20(d)) is better than that of the H_2 controller but worser than that of the H_∞ controller.

The same five experiments are conducted with the designed mixed H_2/H_∞ controller. The obtained simulation results are similar to the H_2 and H_∞ controllers results except that the responsiveness and the queue length needed at the nodes

are different. Only precise explanation of the experiments is given here since it is similar to the explanation given in the previous controller cases.

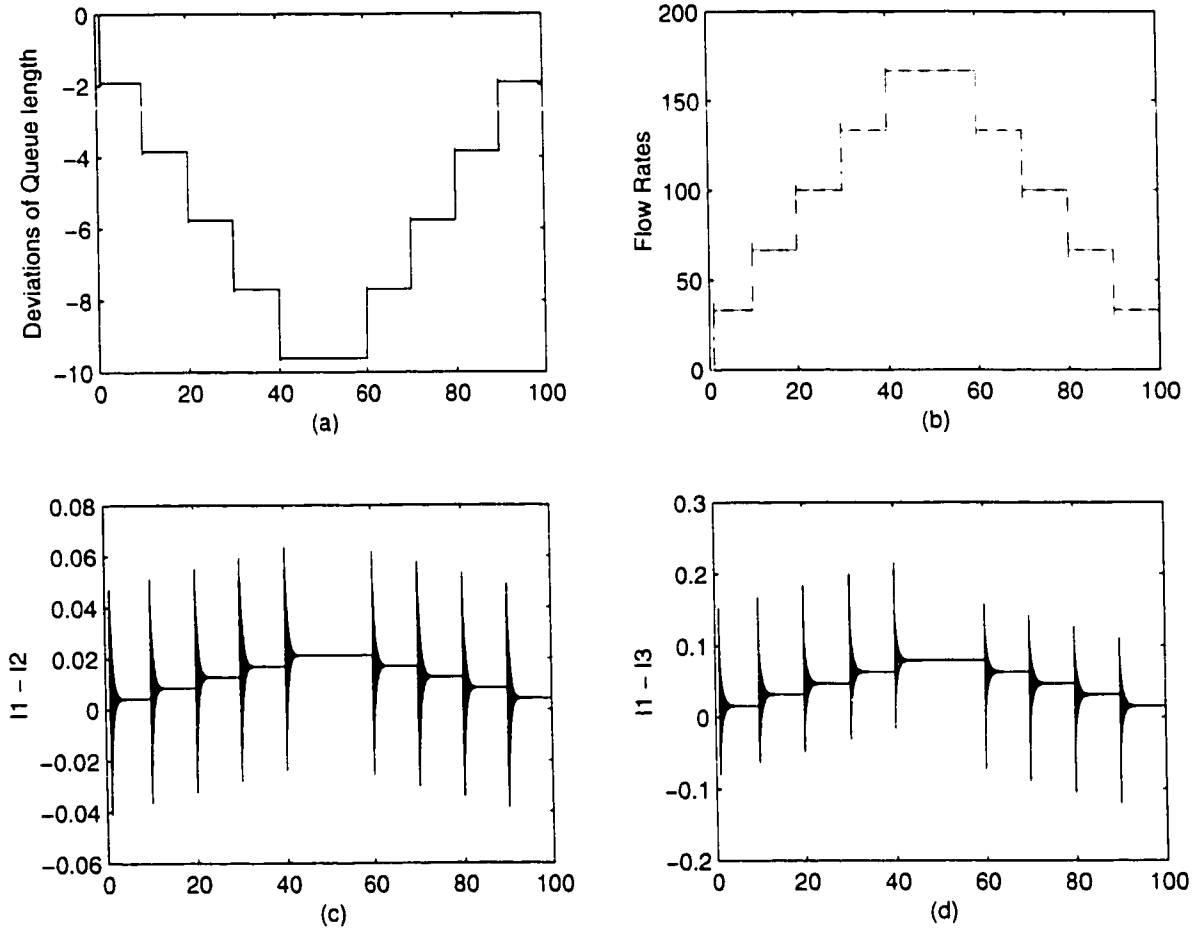


Figure 4.20: The Simulation results of mixed H_2/H_∞ controller

1. **Experiment 1 :** The network simulation results of this experiment with the designed mixed H_2/H_∞ controller are shown in Figure 4.21. The values of the flow rates assigned by the local mixed H_2/H_∞ controllers for $C = 500$ (at $T_s = 50$) are shown in Table 4.17 and the corresponding deviations in queue

Table 4.17: Feedback given by the local mixed H_2/H_∞ controller in Exp. 1 at $T_s = 50$

F/W Rates	f_1	f_2	f_3	f_{11}	f_{12}	f_{13}	f_{21}	f_{22}	f_{23}	f_{31}	f_{32}	f_{33}	ρ
$T_s = 50$	166.7	166.68	166.21	55.58	55.57	55.55	55.57	55.56	55.54	55.54	55.54	55.52	1.0

Table 4.18: Deviations of queue length from the desired value at the different nodes in Exp. 1 at $T_s = 50$

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	−3.20	−3.20	−3.20	−9.61

lengths from the desired value are shown in Table 4.18.

2. **Experiment 2 :** The mixed H_2/H_∞ controller's simulation results are shown in Figure 4.22. These results are similar to the results of the previous experiment except the effect of the noise appears in the queue lengths and in the flow rates.

3. **Experiment 3 :** The mixed H_2/H_∞ controller is simulated and the corresponding simulation results are shown in Figure 4.23. The channel between node 2 and node 4 is broken at the 50^{th} sampling time and node 3 and node 4 is broken at the 55 sampling. Hence, the mixed H_2/H_∞ controller divides the available capacity between the node 1 and node 3 during the 50^{th} to the 55^{th} sampling time and allocates all the capacity to node 1 after the 55^{th} sampling time. The related values are tabulated in Tables 4.19 and 4.20.

Table 4.19: Feedback given by the local mixed H_2/H_∞ controller in Exp. 3 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	250.1	250.1	249.9	83.4	83.4	83.4	0	0	0	83.3	83.3	83.3	1.0
$T_s = 55$	499.9	499.74.1	499.08	165.2	165.2	165.1	0	0	0	0	0	0	1.0

Table 4.20: Deviations of queue length from the desired value at the different nodes in Exp. 3 at critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	−4.81	0	−4.81	−7.62
$T_s = 55$	−9.61	0	0	−41.62

4. **Experiment 4 :** The simulation results obtained with the designed mixed H_2/H_∞ controller are displayed in Figure 4.24. Let us again examine the rates at $T_s = 22$. At this moment the feedback sent to node 1 and node 2 is lost. Hence, node 1 and node 2 continues to send at the rates I_1 ($I_{11} + I_{12} + I_{13} = 66.68$) and I_2 ($I_{21} + I_{22} + I_{23} = 116.69$). So, the remaining capacity ($C - I_1 - I_2 = 116.52$) should be allocated to node 3. From Table 4.21 the value of I_3 can be read as 116.63 which is nearly same as the calculated one. Other events can be interpreted in the same way and all the related values are given in Tables 4.21 and 4.22.

5. **Experiment 5 :** The results of this experiment with the designed mixed

Table 4.21: Feedback given by the local mixed H_2/H_∞ controller in Exp. 4 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	66.68	65.67	66.65	22.23	22.23	22.22	22.23	22.23	22.22	22.22	22.22	22.21	1.0
$T_s = 22$	116.7	116.7	116.63	22.23	22.23	22.22	38.9	38.9	38.89	38.88	38.88	38.87	1.0
$T_s = 45$	191.72	191.69	191.58	63.91	63.89	63.95	38.9	38.9	38.89	63.87	63.87	63.84	1.0
$T_s = 55$	166.7	166.68	166.62	55.58	55.57	55.55	55.53	55.49	55.59	55.55	55.54	55.53	1.0

Table 4.22: Deviations of queue length from the desired value at the different nodes in Exp. 4 at critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−1.28	−1.28	−1.28	−3.84
$T_s = 22$	−1.28	−2.24	−2.24	−7.37
$T_s = 45$	−3.69	−2.24	−3.68	−12.02
$T_s = 55$	−3.20	−3.20	−3.20	−9.61

H_2/H_∞ controller using inputs in Figure 4.4 and 4.5 are shown in Figure 4.25 and 4.26 respectively. From these figures it can be easily concluded that the mixed H_2/H_∞ controller, as other controllers, is also efficiently managing the situation.

Table 4.23: Feedback given by the local mixed H_2/H_∞ controller in Exp. 5 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	65.71	65.33	65.65	21.92	21.92	21.23	21.97	22.67	24.94	22.22	22.24	21.40	1.0
$T_s = 45$	428.8	428.9	427.7	141	142.8	142.3	21.91	22.67	24.94	0	0	0	1.0
$T_s = 55$	251.30	247.97	248.95	85.06	83.63	84.40	83.03	83.8	83.40	0	0	0	1.0

Table 4.24: Deviations of queue length from the desired value at the different nodes in Exp. 5 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−1.27	−1.31	−1.18	−3.65
$T_s = 45$	−8.23	−1.32	0	−34.88
$T_s = 55$	−4.78	−4.76	0	−17.65

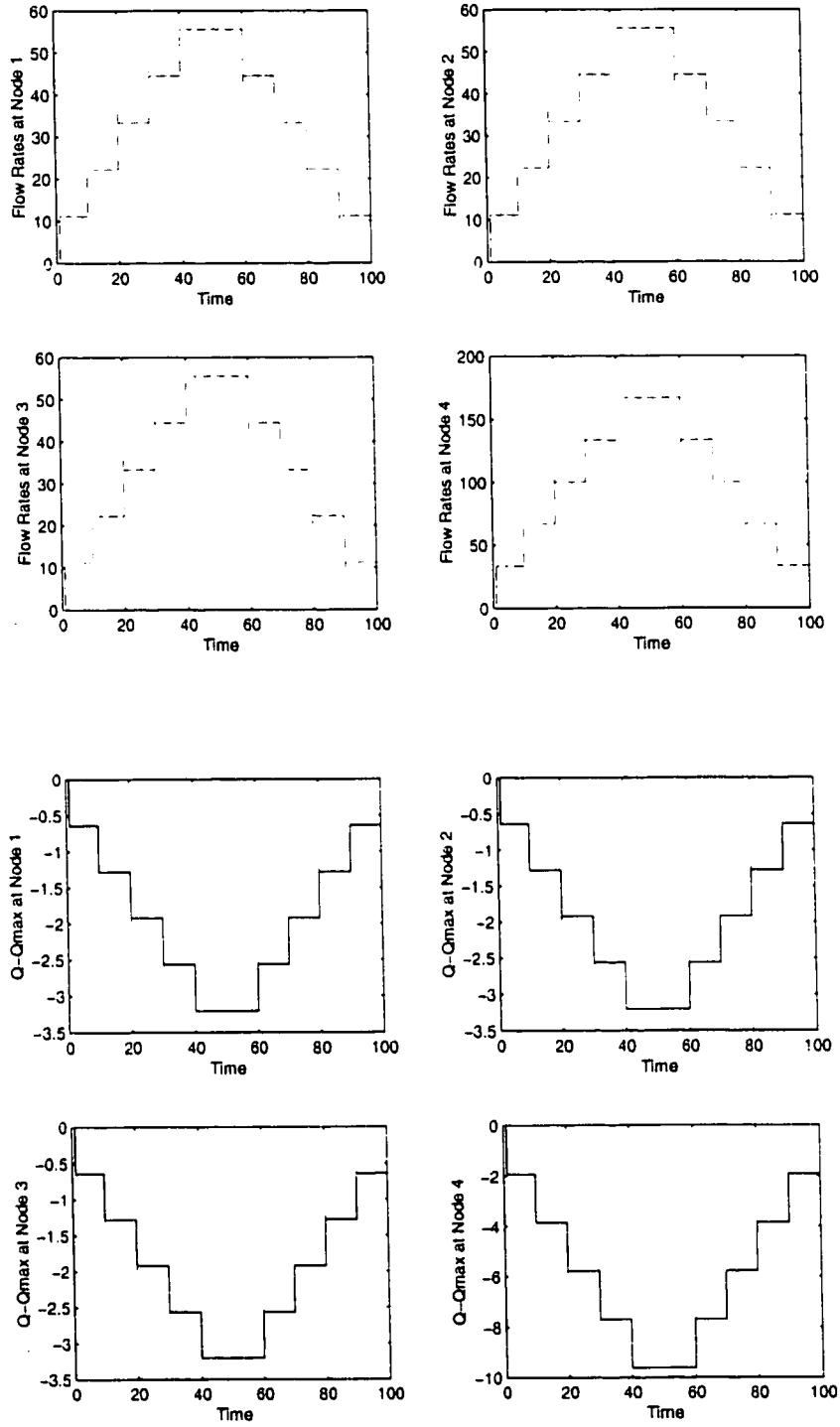


Figure 4.21: The Mixed H_2/H_∞ Simulation results of Exp. 1

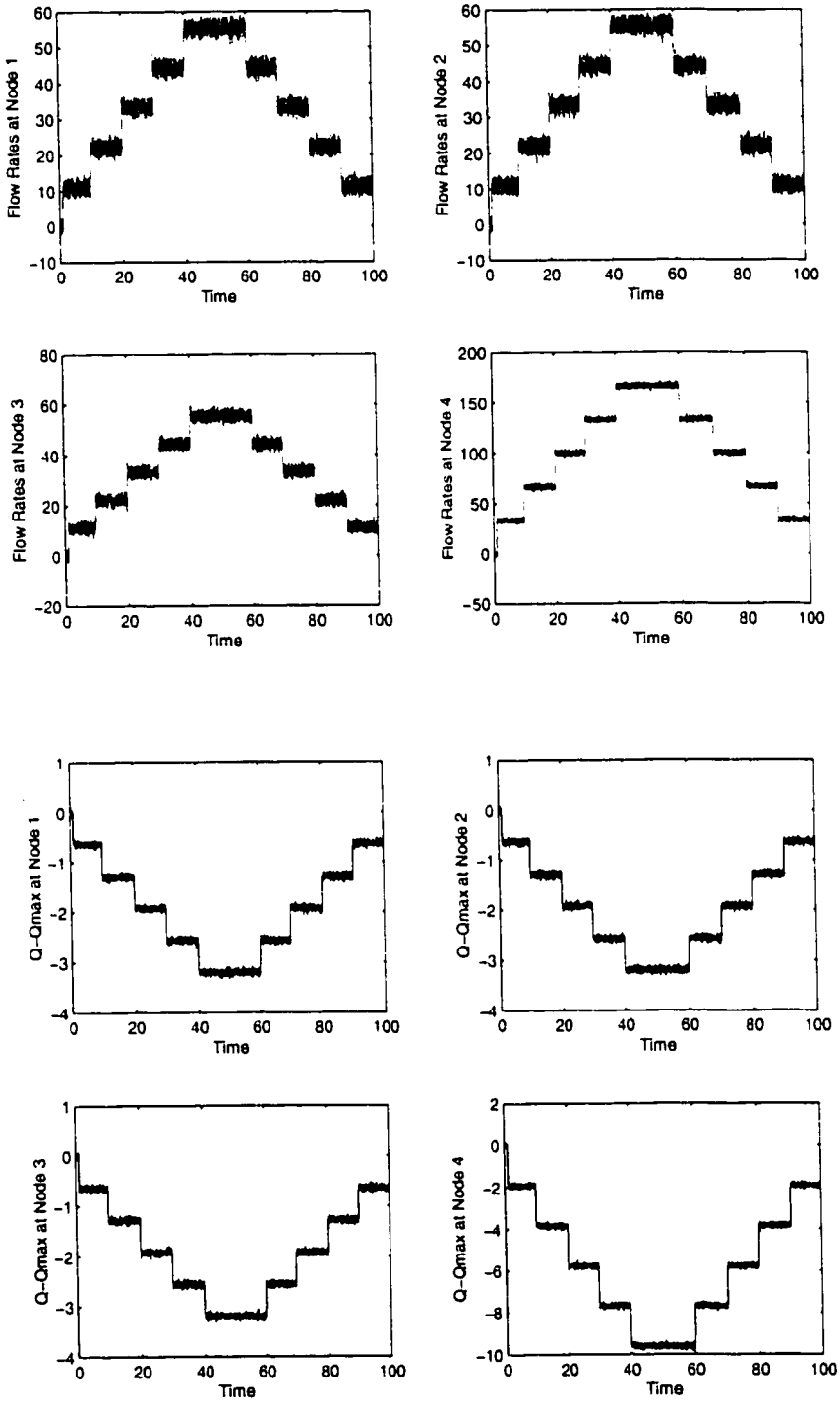


Figure 4.22: The Mixed H_2/H_∞ Simulation results of Exp. 2

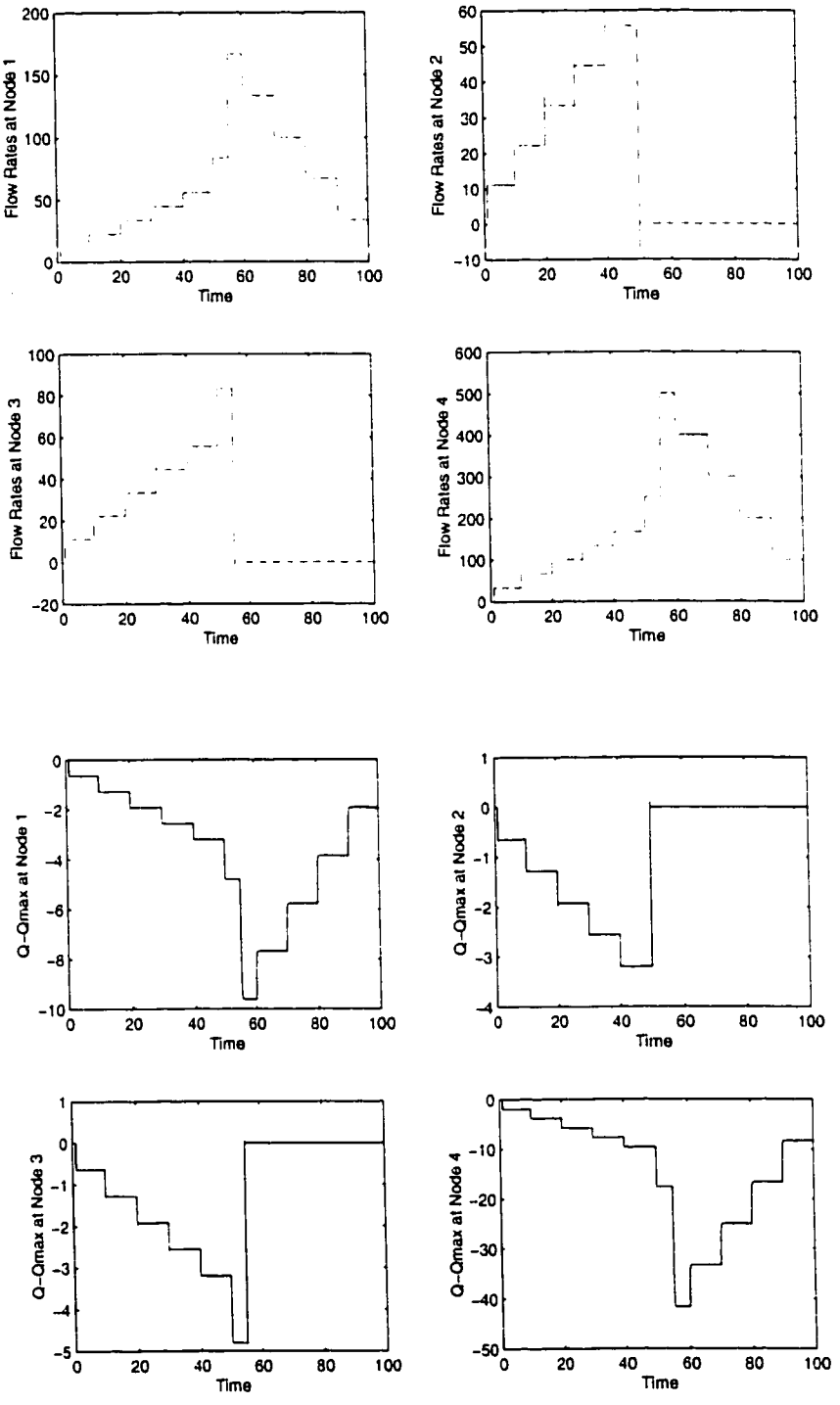


Figure 4.23: The Mixed H_2/H_∞ Simulation results of Exp. 3

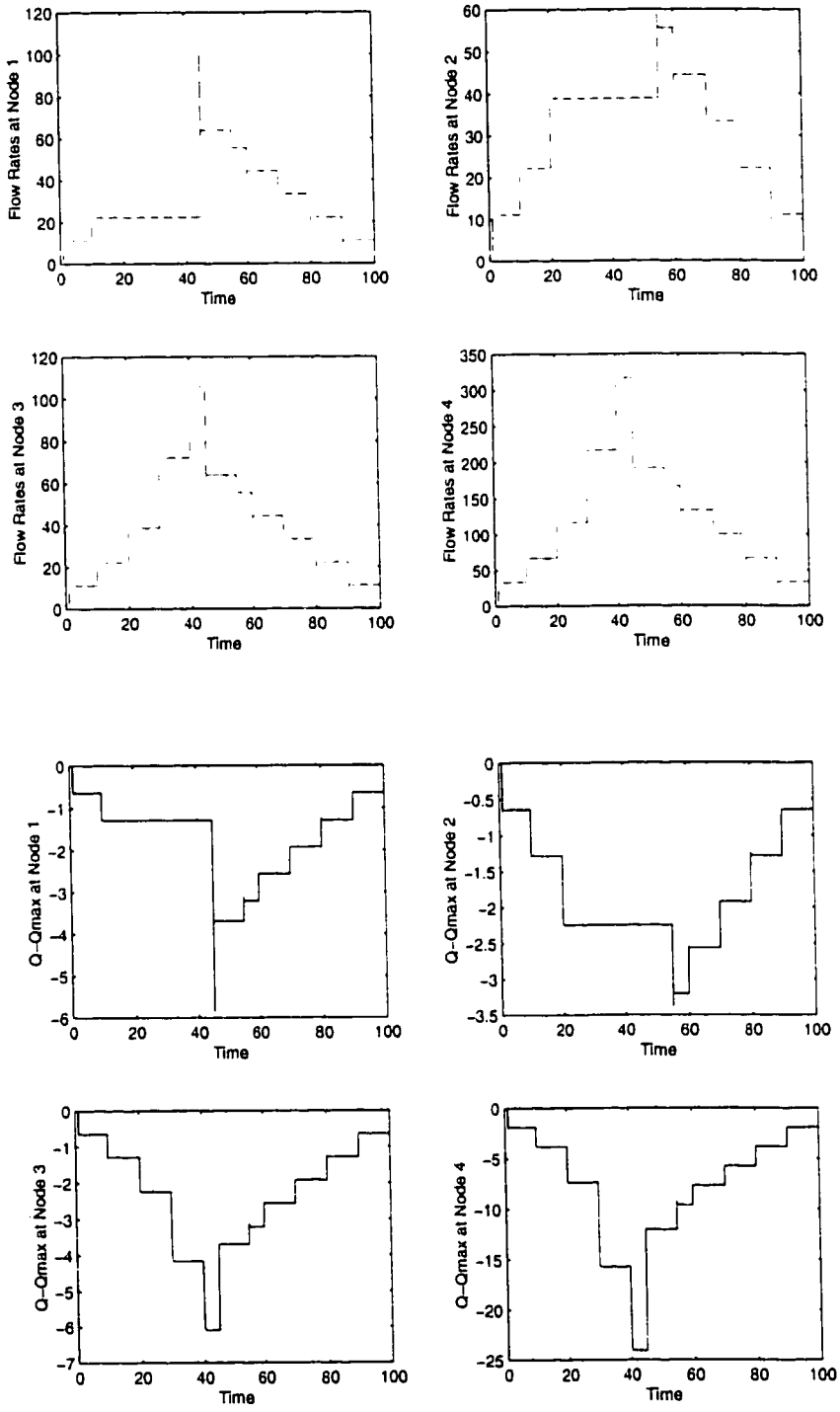


Figure 4.24: The Mixed H_2/H_∞ Simulation results of Exp. 4

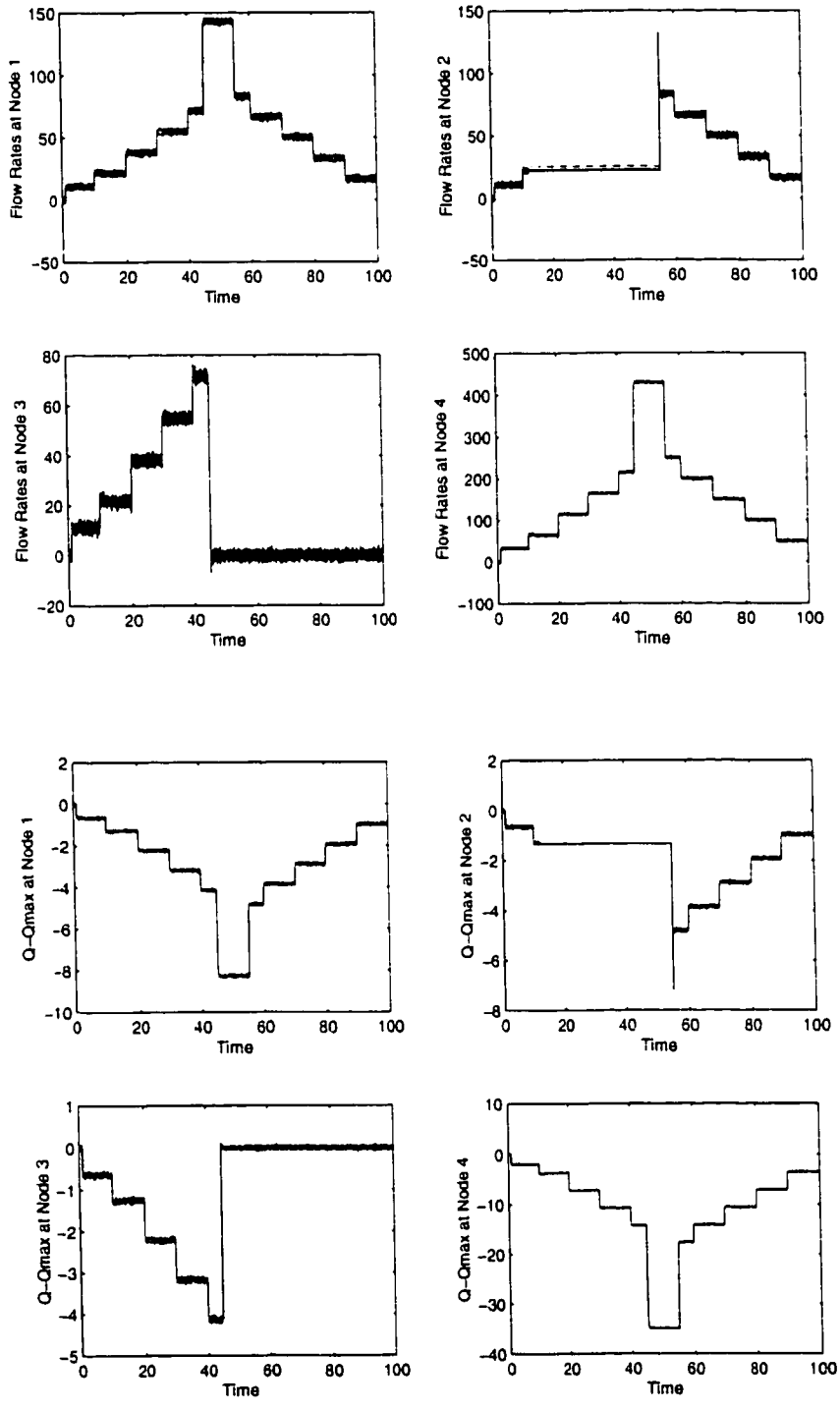


Figure 4.25: The Mixed H_2/H_∞ Simulation results of Exp. 5

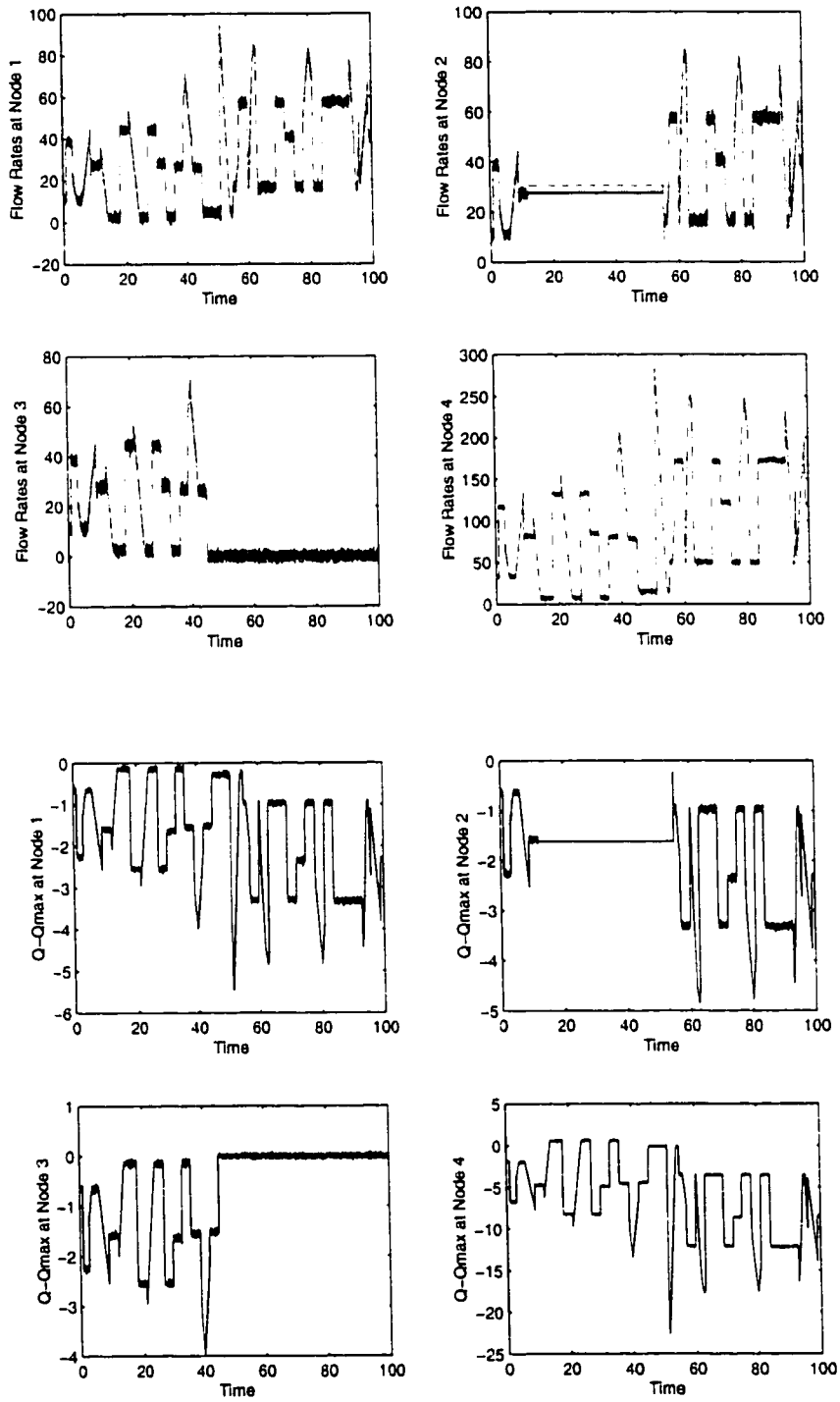


Figure 4.26: The Mixed H_2/H_∞ Simulation results of Exp. 5 with the random input

4.4.4 Adaptive Controller

The controller described by Eq. (3.40) is used as the local controller at the node of the network shown in Figure 4.1. The initial values of the controller parameters are $k_1 = 0$ (since, $I(n-1) = 0$) and $k_2 = k_3 = \frac{1}{3}$ (as the number of sources is 3). The simulation results are shown in Figure 4.27. From Figure 4.27(a) we can see that the queue length in the steady state reaches zero. This is true since the controller parameters are chosen such that the queue length reaches a constant value. Another point worth noting is that the controller is always fair as the same feedback I is given to all the sources. For this reason the plots of difference between flow rates is omitted. The utilization of the network is naturally one. These results should not be surprising as the controller parameters are chosen to achieve this kind of performance. The actual performance of adaptive controller will be evident only after simulating the controller in the networked environment under different situations. So the same five experiments are repeated with adaptive controller.

1. **Experiment 1 :** The network simulation results for this case with the designed adaptive controller are shown in Figure 4.29. These results are similar to the single node case. The values of the flow rates assigned by the local controllers for $C = 500$ are shown in Table 4.25. The steady state queue lengths are zero. Note that this is an ideal case and initial parameters of adaptive

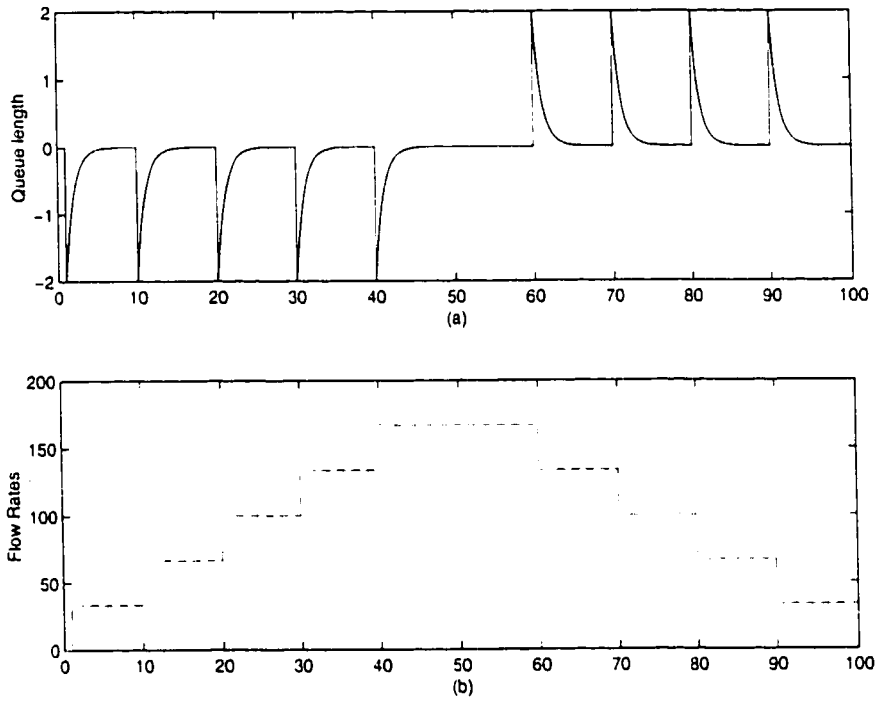


Figure 4.27: The simulation results of adaptive controller

Table 4.25: Feedback given by the local adaptive controller in Exp. 1 at the critical times

F/W Rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	166.67	166.67	166.67	55.55	55.55	55.55	55.55	55.55	55.55	55.55	55.55	55.55	1.0

controller are chosen to attain this kind of nice performance. Since the state of the network has not changed the value of k_2 remains constant at 0.333 (see Figure 4.28(a)).

2. **Experiment 2 :** The adaptive controller 's simulation results of Exp. 2 are shown in Figure 4.30. As in the case of previously designed controllers the results are analogous to Exp. 1 results except that the effect of the noise appears on the queue lengths and the flow rates.

3. **Experiment 3 :** The adaptive controller 's simulation results for this experiment are shown in Figure 4.31. At $T_s = 50$, the channel between node 2 and node 4 is broken. Hence, the controller has to divide the available capacity between the node 1 and node 3. From Table 4.11 the values of I_1 (= I_2) is 192.57. Thus, it is obvious that the controller is unable to achieve the efficiency immediately. Nevertheless, the controller attains efficiency after the parameter k_2 is completely tuned to adapt the changes in the network. Figure 4.28(b) shows the changes in the value of k_2. An analogous effects can be seen at node 3, at 55^{th} sampling time. The related values are tabulated in Tables

Table 4.26: Feedback given by the local adaptive controller in Exp. 3 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 50$	192.57	192.57	192.57	64.33	64.33	64.33	0	0	0	64.33	64.33	64.33	0.77
$T_s = 55$	311.02	311.02	311.02	104.03	104.03	104.03	0	0	0	0	0	0	0.62

Table 4.27: Deviations of queue length from the desired value at the different nodes in Exp. 3 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 50$	−0.41	0	0.41	−67.34
$T_s = 55$	−1.074	0	0	−174.14

4.26 and 4.27. These results are inferior to the results obtained with the robust controllers in the preceding sections. However, the adaptive controller's results are more appreciated when compared with the results of the controller with no adaption (without tuning k_2) shown in Figure 4.32.

4. **Experiment 4 :** The simulation results obtained for this experiment with the designed adaptive controller are displayed in Figure 4.33. Let us examine the rates at $T_s = 22$. At this moment the feedback sent to node 1 and node 2 is lost. Hence, node 1 and node 2 continues to send at the rates I_1 ($I_{11} + I_{12} + I_{13} = 66.66$) and I_2 ($I_{21} + I_{22} + I_{23} = 116.69$). So, the remaining capacity ($C - I_1 - I_2 = 117.60$) should be allocated to node 3. From Table 4.28 the value of I_3 can be read as 116.7 which is approximately same as the calculated one. The value of k_2 is plotted in Figure 4.28(c). From this figure, we can see that the value of k_2 started decreasing after $T_s = 55$. This is because the network has reached again the original state. For comparison, the simulation

Table 4.28: Feedback given by the local adaptive controller in Exp. 4 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	66.59	66.59	66.59	22.22	22.22	22.22	22.21	22.21	22.21	22.21	22.21	22.21	1.0
$T_s = 22$	116.7	116.7	116.7	22.22	22.22	22.22	38.58	38.58	38.58	38.93	38.93	38.93	0.99
$T_s = 45$	192.2	192.2	192.2	64.36	64.36	64.36	38.58	38.58	38.58	63.72	63.72	63.72	0.99
$T_s = 55$	166.05	166.05	166.05	55.29	55.29	55.29	56.09	56.09	56.09	55.28	55.28	55.28	0.99

Table 4.29: Deviations of queue length from the desired value at the different nodes in Exp. 4 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	-0.08	-0.03	-0.03	0.22
$T_s = 22$	-0.08	-0.19	-0.08	-29.77
$T_s = 45$	-1.3	-0.19	1.04	-70.30
$T_s = 55$	-0.676	0.6	0.22	-65.17

results when there is no tuning is shown in Figure 4.34.

5. **Experiment 5 :** The results of this experiment with the designed adaptive controller using both stair and random inputs are shown in Figure 4.35 and 4.36. The changes in the values of k_2 (for stair input) are shown in Figure 4.28(d). From these figures it can be inferred that the adaptive controller's responsiveness is very less. Consequently, the adaptive controller needs more queue length (buffer space) to efficiently utilize the available bandwidth.

Table 4.30: Feedback given by the local adaptive controller in Exp. 5 at the critical times

F/W rates	I_1	I_2	I_3	I_{11}	I_{12}	I_{13}	I_{21}	I_{22}	I_{23}	I_{31}	I_{32}	I_{33}	ρ
$T_s = 12$	66.38	66.38	66.66	22.72	21.77	24.06	22.57	21.18	21.36	23.26	23.26	20.98	0.99
$T_s = 45$	397.89	397.89	397.47	135.42	131.36	132.94	22.57	21.18	21.36	0	0	0	1.0
$T_s = 55$	251.77	251.77	251.10	84.68	84.23	83.57	84.52	84.57	85.85	0	0	0	1.0

Table 4.31: Deviations of queue length from the desired value at the different nodes in Exp. 5 at the critical times

$Q - Q_{desired}$	q_1	q_2	q_3	q_4
$T_s = 12$	−0.03	−0.0285	−0.0375	−0.2438
$T_s = 45$	1.52	−2.94	0	−178.5
$T_s = 55$	1.52	−2.94	0	−142.34

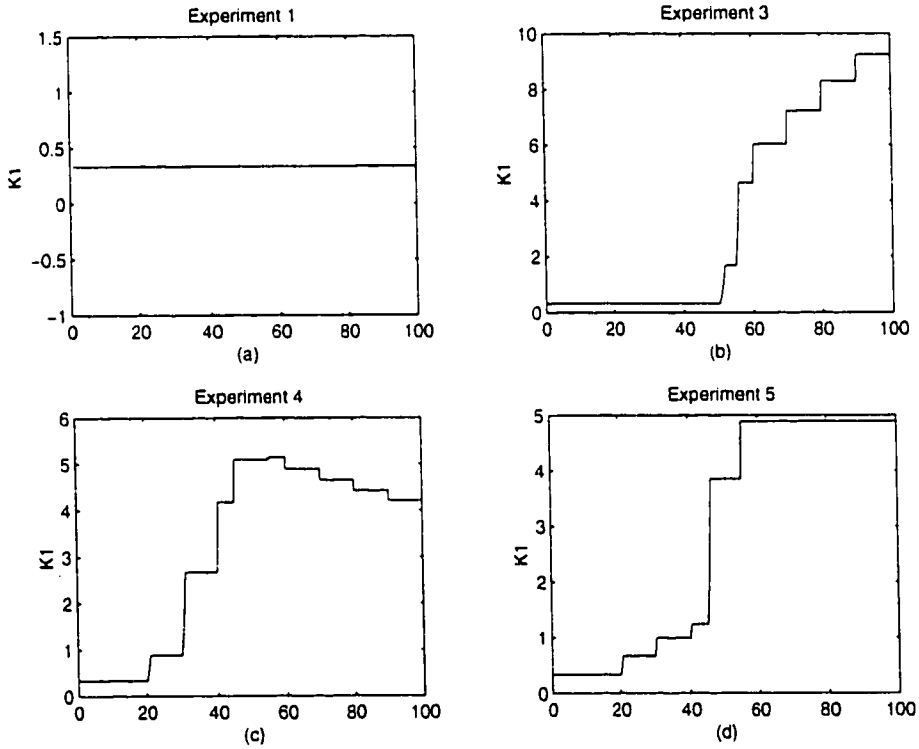


Figure 4.28: The values of k_2 in different experiments

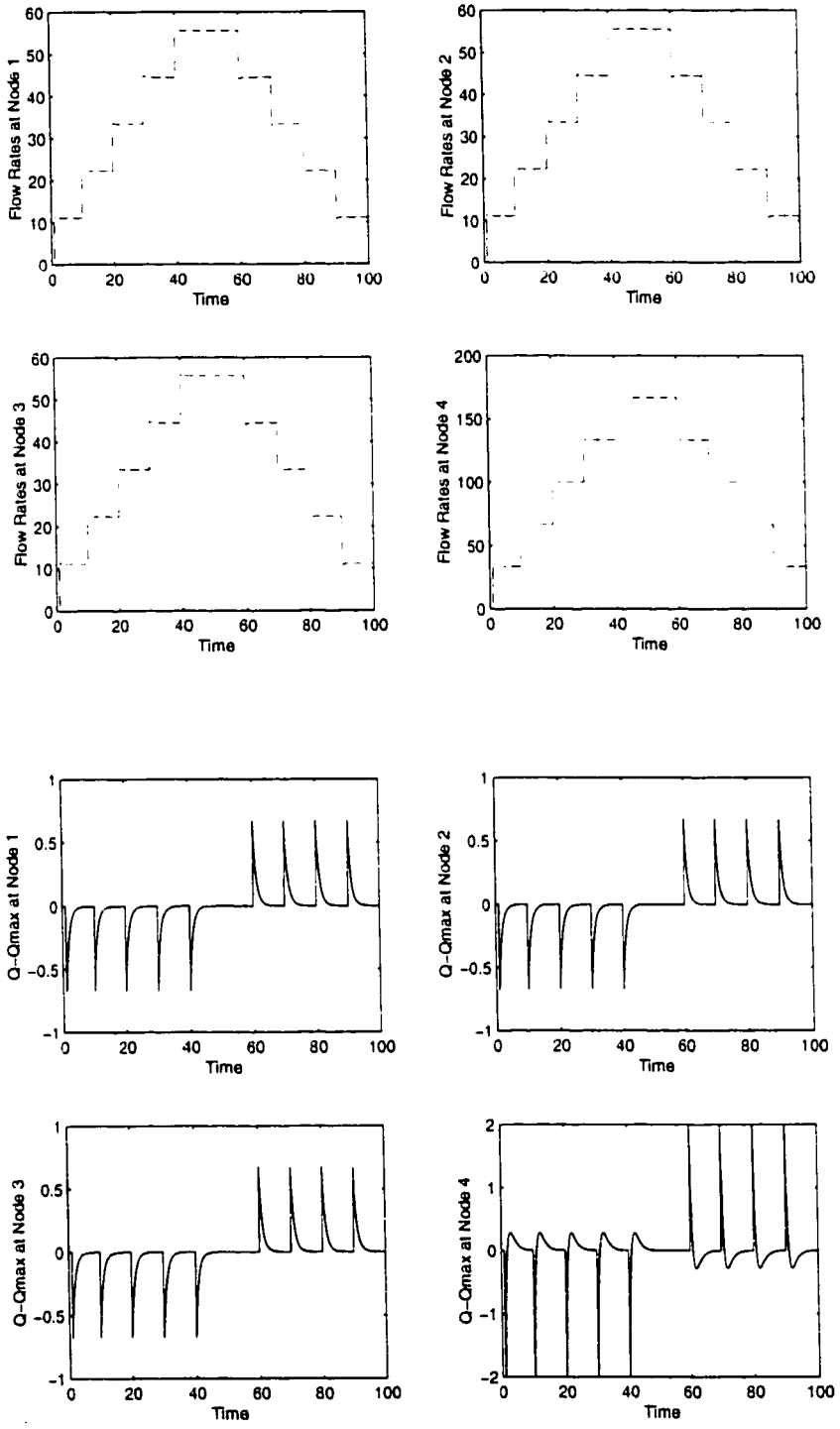


Figure 4.29: Adaptive Controller's Simulation results for Exp. 1

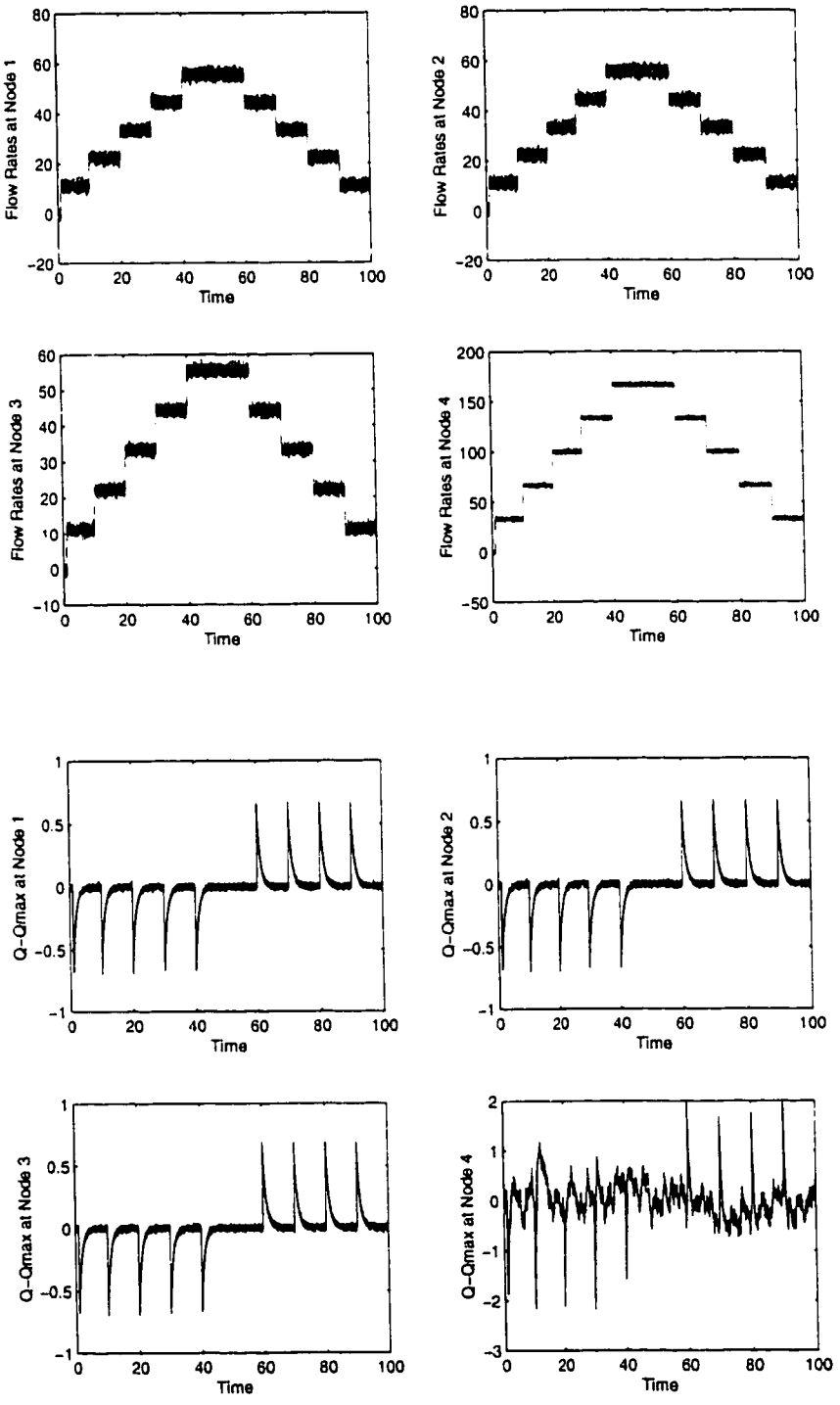


Figure 4.30: Adaptive Controller's Simulation results for Exp. 2

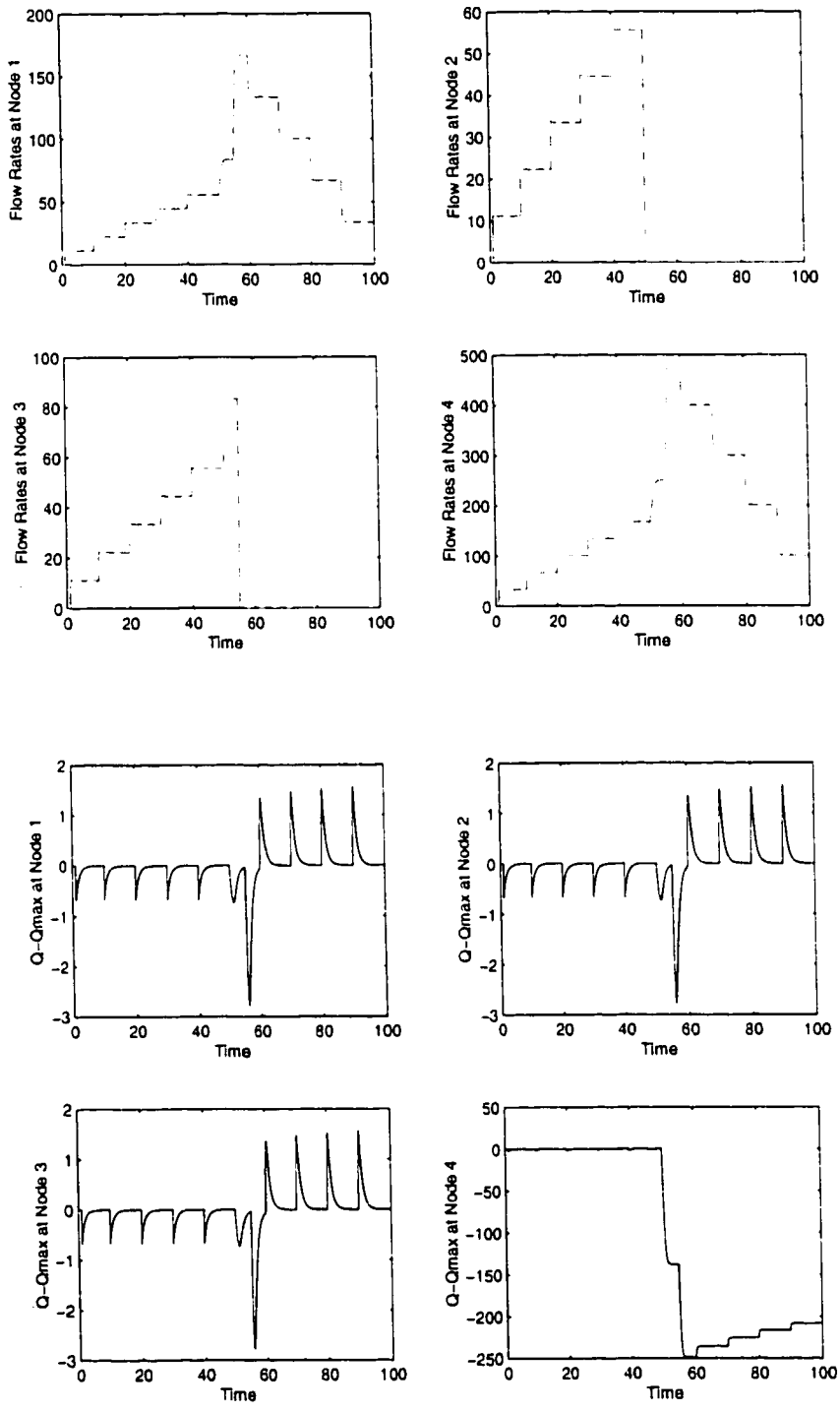


Figure 4.31: Adaptive Controller's Simulation results for Exp. 3

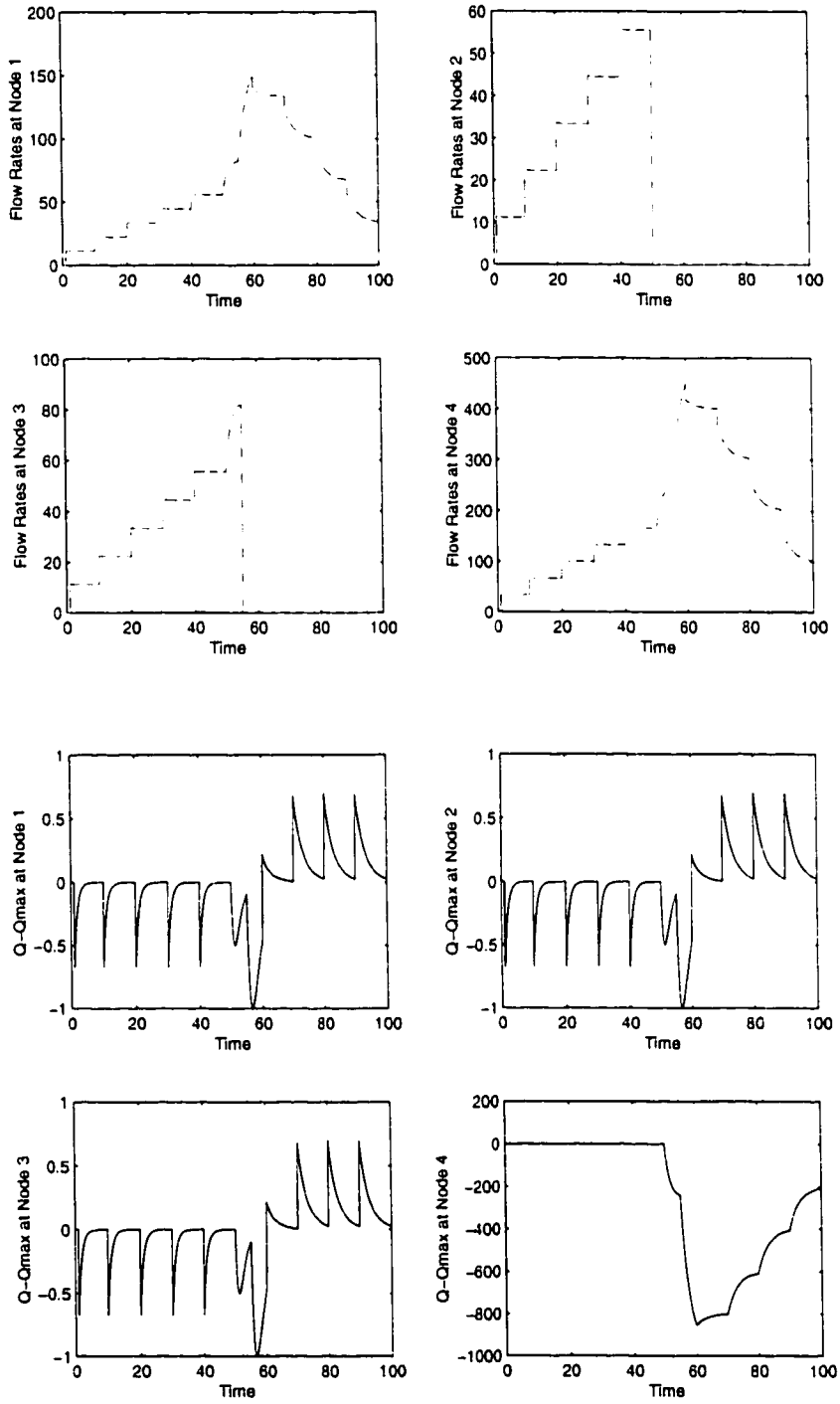


Figure 4.32: Adaptive Controller's Simulation results for Exp. 3 when k_2 is not tuned

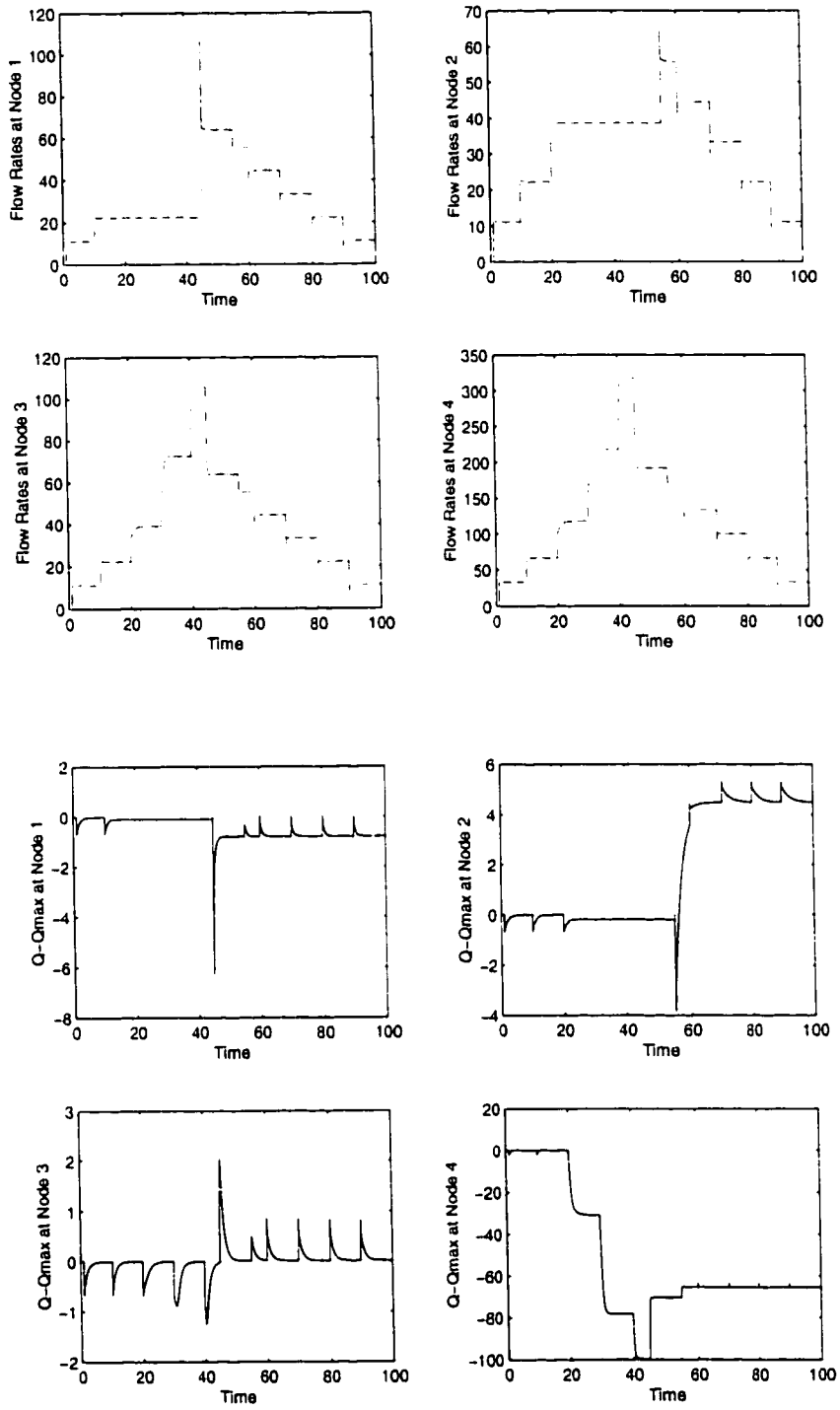


Figure 4.33: Adaptive Controller's Simulation results for Exp. 4

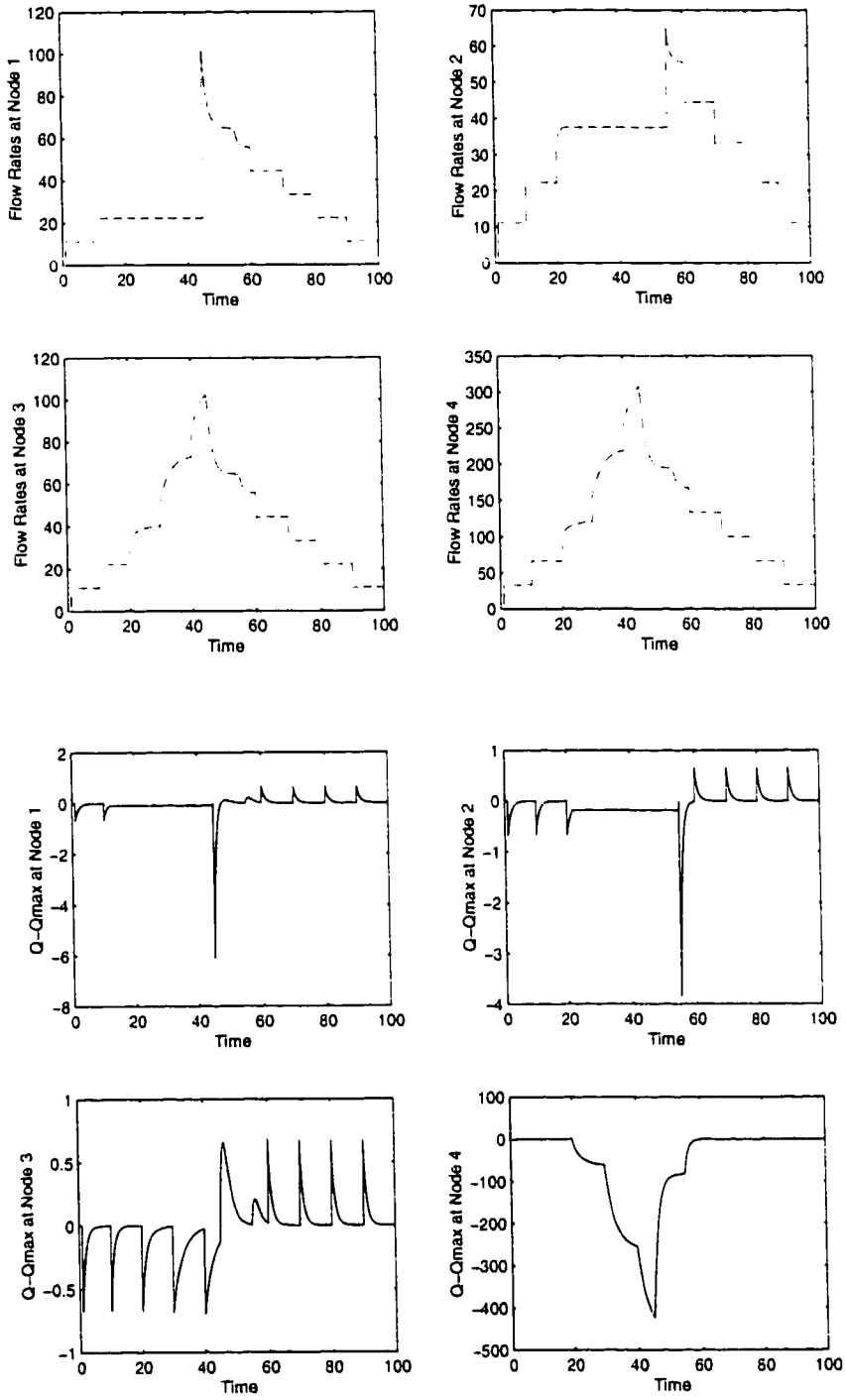


Figure 4.34: Adaptive Controller's Simulation results for Exp. 4 when k_2 is not tuned

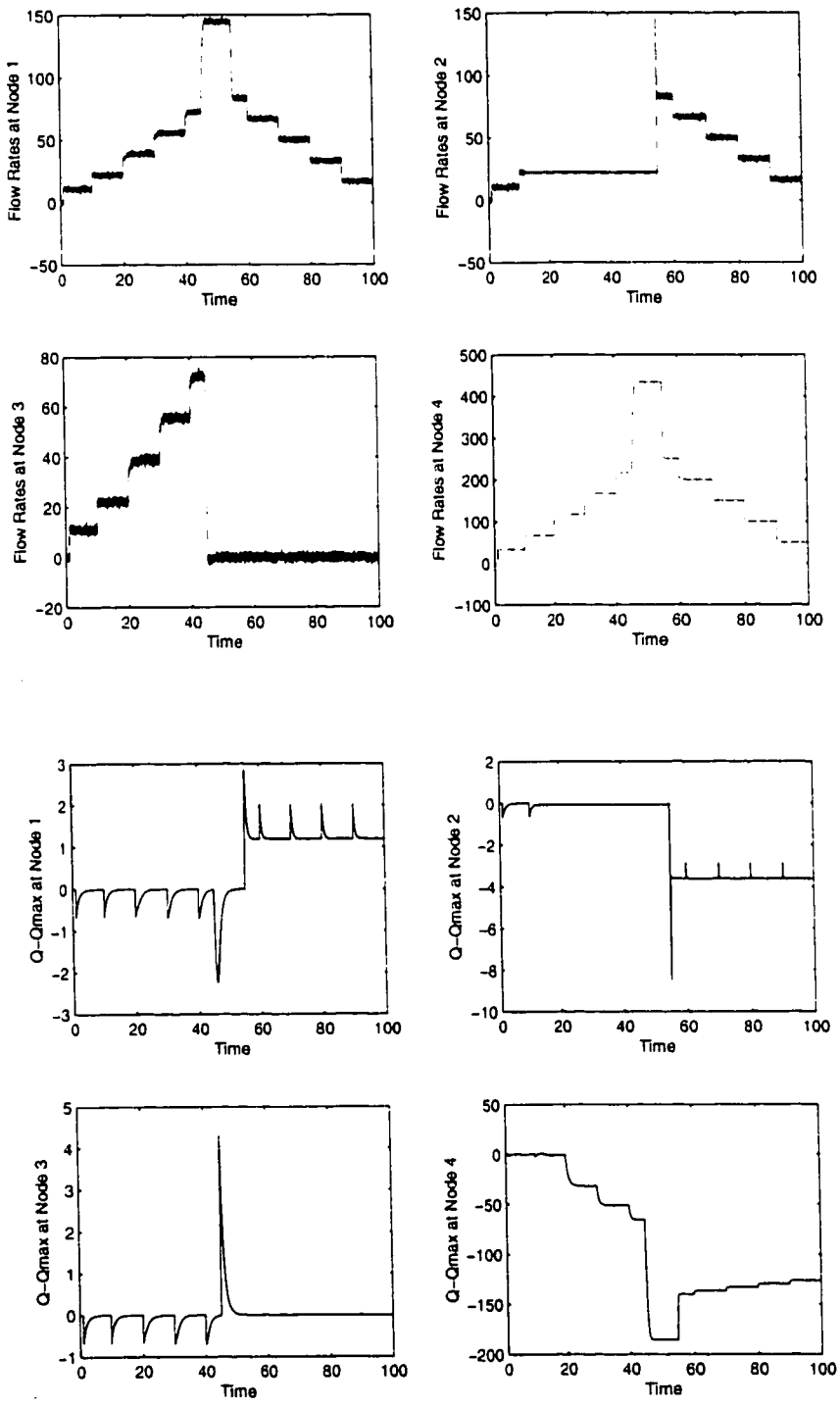


Figure 4.35: Adaptive Controller's Simulation results for Exp. 5

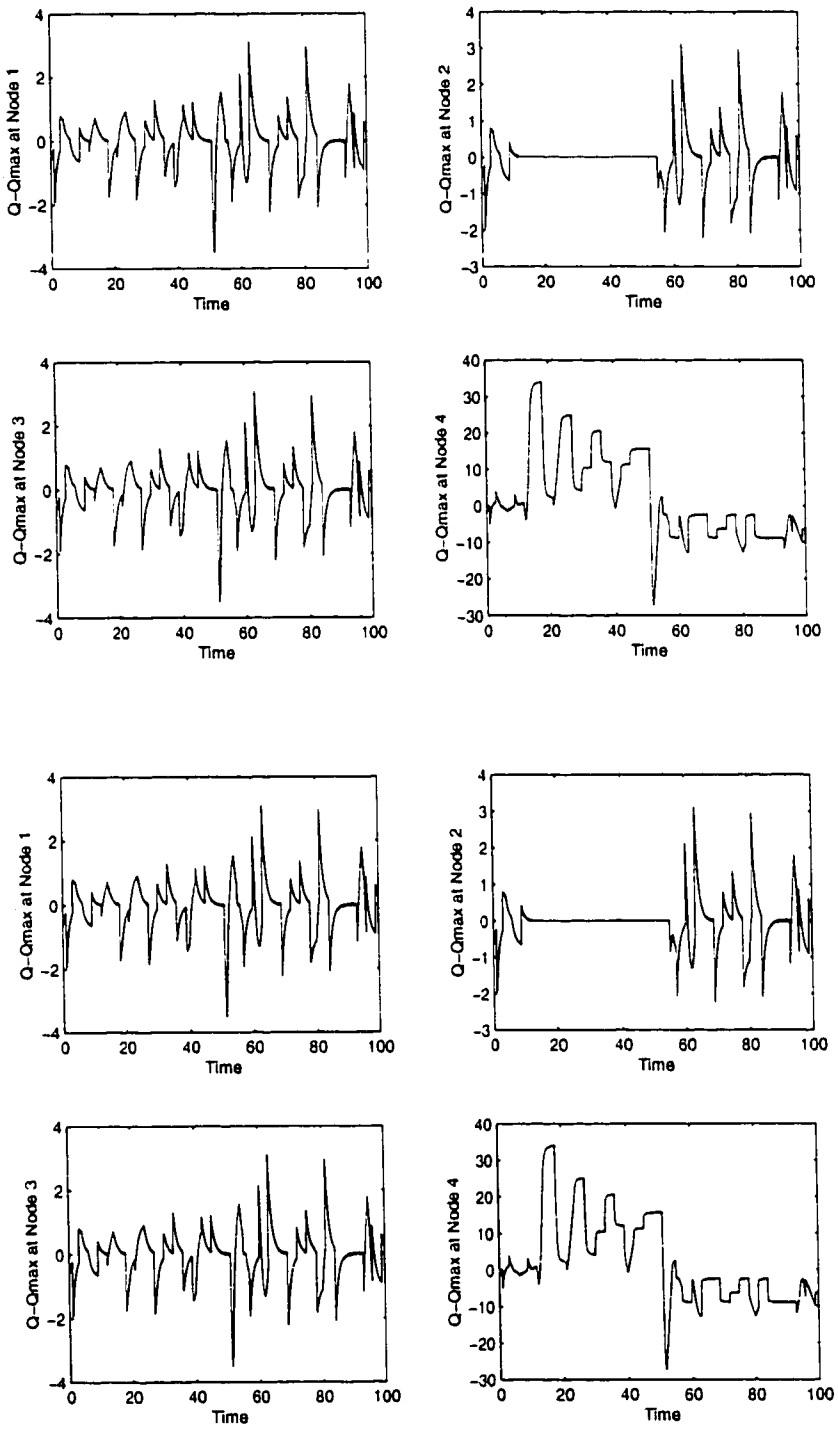


Figure 4.36: Adaptive Controller's Simulation results for Exp. 5 with random input

4.4.5 Comparison of Robust Controller's Results with the Available Methods In the Literature

In this section we will compare the proportional (P) compensator and the Proportional Integrator (PI) compensator described in [25] with our methods. The model for the single node network of Fig. 4.1 is drawn in Fig. 4.37.

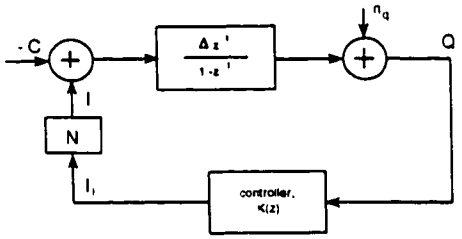


Figure 4.37: Model for the network with N sources and one switch

The forward path of the figure is the queue model. Based on queue size, controller $K(z)$ calculates the desired rate to be fed to the sources. Since all the sources are asked to send at the same rate, the total input rate arriving at the switch is N times the rate fedback. A white gaussian noise (n_q) is added to take care of the modeling error.

Proportional Compensator

The simple choice for $K(z)$ is to use a proportional controller, that is, the of rate of the i^{th} source (I_i) is taken as proportional to the instantaneous queue size $Q(z)$. The controller is implemented by the following equation.

$$I_i(n) = L - KQ(n). \tag{4.10}$$

Where K is a proportionality constant and L is a necessary constant to make sure that the resulting rate is a positive value. Using Nyquist plot the condition for the stability of the closed loop system can be obtained as $KN\Delta < 2$ [25]. Similarly the maximum delay for which the linearized system will remain stable can be found as $\left(\frac{PM(rads)}{\omega_c}\right)\Delta$ [25]. Where $PM(rads)$ is the phase margin in radians. Thus, increasing Δ, while keeping $KN\Delta$ constant, would make the system robust to larger delays. But increasing Δ lengthens the time it takes for the network to respond to the traffic conditions which will in-turn increase the queue length requirements.

The results obtained for single node network with $KN\Delta = 1$ is shown in Fig. 4.38. From this figure we can infer that proportional controller needs much more

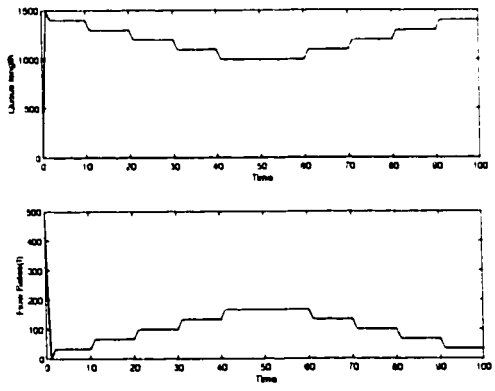


Figure 4.38: Proportional controller results for single node network

queue length than that of robust controllers. However, in other aspects such as fairness and network utilization, proportional controller is doing well. It should be noted that these results are obtained for an ideal case. Thus, the experiment 5 is conducted with this compensator and corresponding results are shown in Fig. 4.39. These figures reveal that the controller is unstable in the network environment. This is true since the locally chosen proportionality gain has been unable to preserve the globally stability. These results can be improved by dynamically tuning controller parameters.

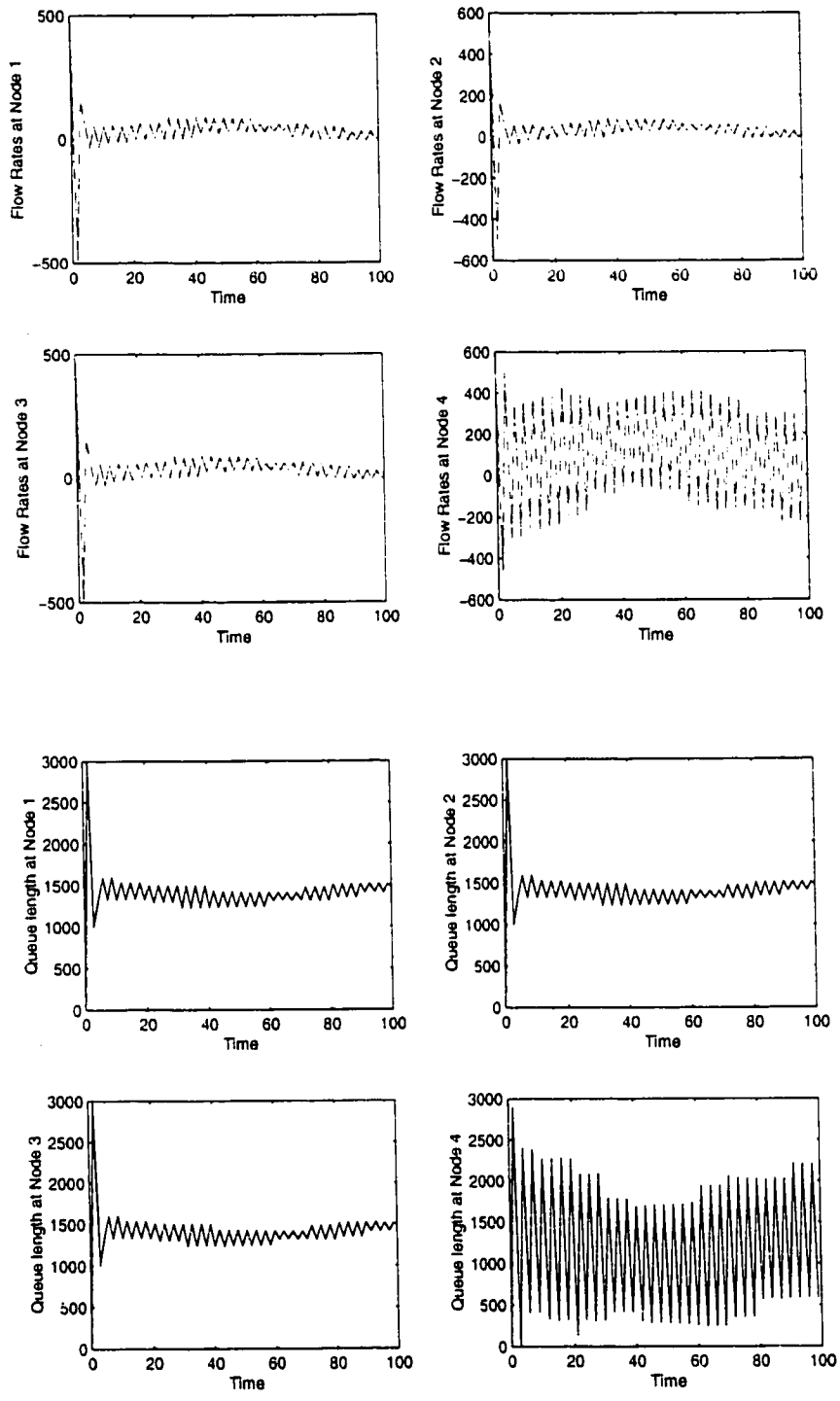


Figure 4.39: Proportional controller results for experiment 5

Lag Compensator

A method for keeping the steady state queue lengths small is to design a controller with an integrator. A standard compensator which accomplices this is a PI or lag compensator. This compensator has a form:

$$K(z) = \left(\frac{-2K}{1+a}\right)\left(\frac{z-a}{z-1}\right). \tag{4.11}$$

This can be thought of a proportional term (K) in series with a lag term, which has unit frequency gain. The design procedure of lag compensator involves designing proportional controller (K) to achieve adequate robustness to delays and choosing 'a' to make steady state queue length zero without loosing designed robustness.

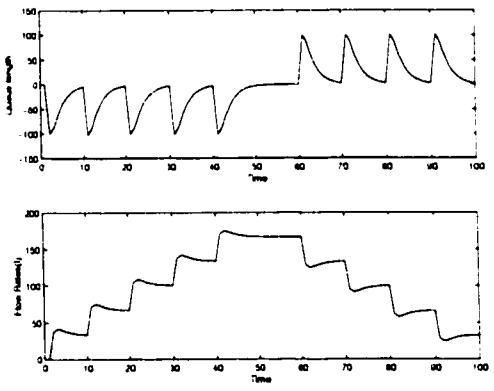


Figure 4.40: PI controller results for single node network

The results obtained for $KN\Delta = 0.8$ is shown in Fig. 4.40. As expected lag compensator is driving the queue length towards zero. This is accomplished by ini-

tially driving the rates of the sources below steady state value, so that queue empties out. Thus, some efficiency is sacrificed for low steady state queue length. Another problem with lag compensator is that if the queue is empty and the incoming rates are less than the available capacity the lag compensator will not increase the rates. These facts can be observed in the experiment 5 results shown in Fig. 4.41. Recall that Exp. 5 is a comprehensive experiment. A compensator which remained stable in Exp. 5 is likely to be stable under all conditions. From Fig. 4.41, it can be seen that unlike proportional controller the PI controller is stable. The responsiveness of this controller is poor which is sacrificied to achieve zero queue lengths in the steady state. However, the queue lengths in the transient periods are very large. This means that eventhough steady state queue length is zero, PI controller needs large buffer space in order to avoid cell losses in transient periods. Thus, even the purpose of sacrificing responsiveness is not achieved

Comparing the results of the robust controllers with these results one can easily deduce that performance of the robust controller's is much better in terms of all the designed specifications and in all circumstances.

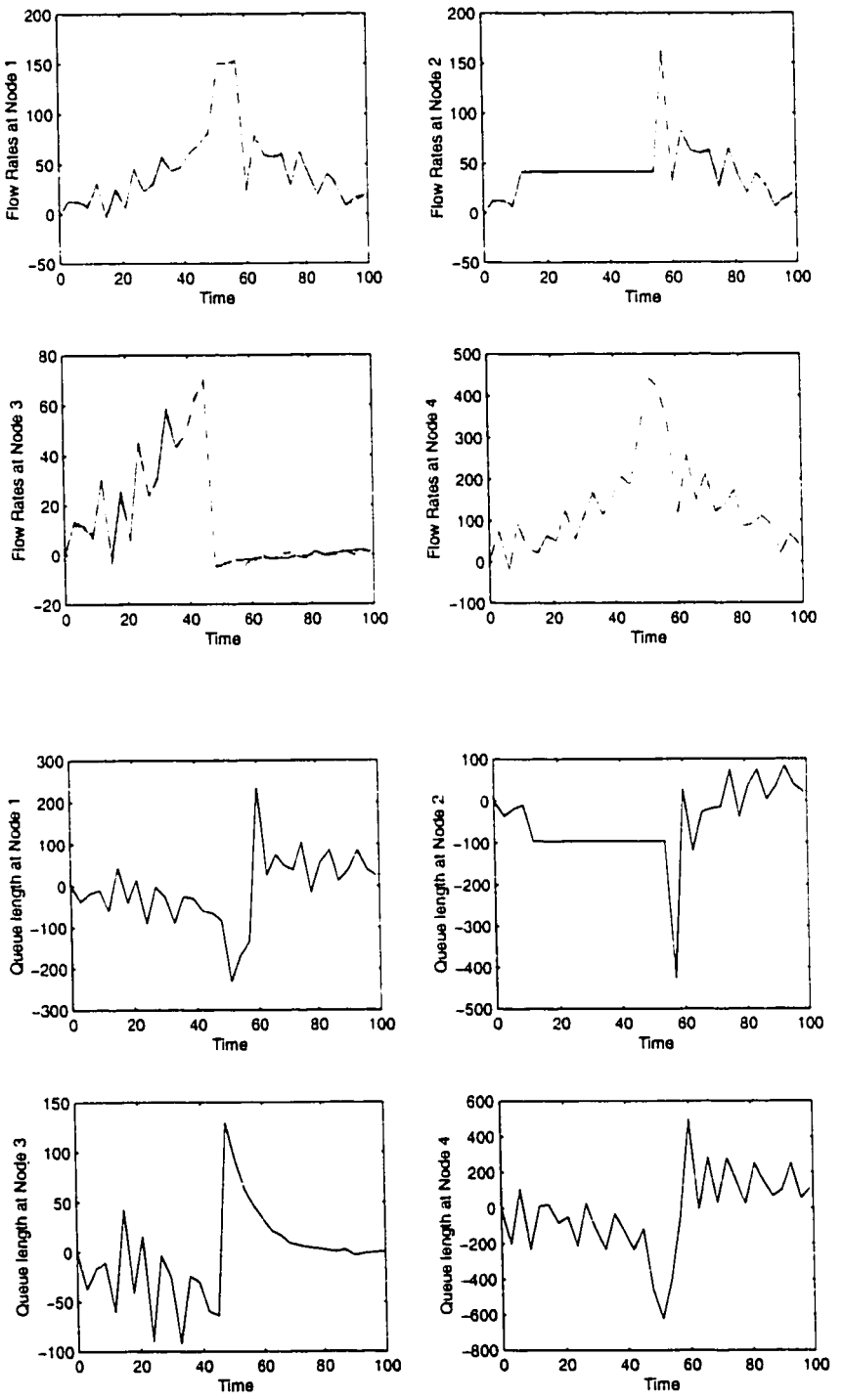


Figure 4.41: Lag compensator results for experiment 5

Chapter 5

DESIGN OF A DATALINK LAYER INTERFACE LIBRARY

5.1 Introduction

In the preceding chapters the design and simulation of congestion control algorithms is discussed. In this chapter, we summarize the Datalink Layer Interface (DLI) library which is useful in testing and implementing congestion control algorithms. The DLI library provides an easy to use interface and implements some of the functions of the Logical Link Control (LLC) sublayer of the DataLink Layer (DLL). Its design is inspired by the well-known *libpcap* library used for network monitoring programs.The internal structure of the library is significantly different in BPF and DLPI. Therefore, we will discuss both cases separately in the earlier sections of the

chapter and the developed library in the later sections. An example procedure for the real-time simulation of congestion control algorithms is given in the final section.

5.2 BSD Packet Filter (BPF)

BSD and many other Berkeley-derived implementations support BPF, the BSD packet filter. BPF is basically a pseudo device: it is not a driver for a particular network interface. However, network interfaces with BPF compliant drivers can be *attached* to it. BPF has two main components: the network tap and the packet filter.

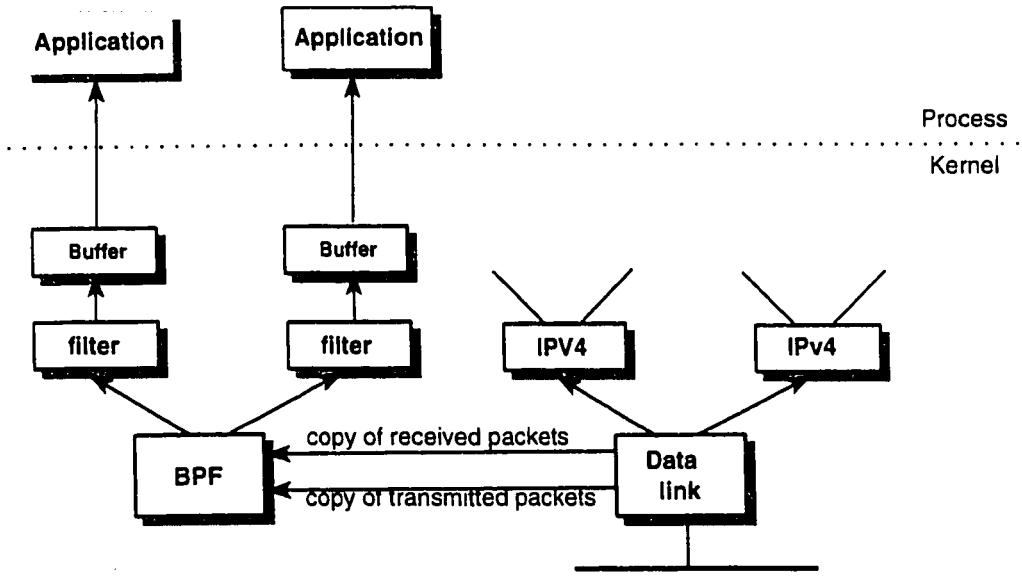


Figure 5.1: Packet capture using BPF

The Network Tap

The network tap of BPF collects copies of packets from network device drivers and delivers them to listening applications. The packet filter of BPF decides whether a packet should be accepted and, if so, how much of it should be copied to listening applications.

Figure 5.1 illustrates how BPF works. When a packet arrives at a network interface, the link level device driver calls BPF, if BPF is listening. Otherwise datalink driver passes packet to the system protocol stack as usual. BPF feeds received packet to each participating process's filter. For each filter that accepts the packet, BPF copies the requested amount of data to the buffer associated with that filter. The device driver then regains the control. If the packet is not addressed to the local host, the driver returns from the interrupt. Otherwise, normal protocol processing proceeds.

Sometimes, applications need to look at each packet, and the time between packets is merely a few micro seconds, and it is not possible to do a read system call per packet. So, BPF collects the data from several packets and returns it as a unit when a monitoring application does a read. To maintain packet boundaries, BPF encapsulates the captured data from each packet with a header that includes the time stamp, length and offsets for data alignment [39, 40].

The Packet Filter

In many situations, network monitors need only a small subset of network traffic, so performance can be drastically improved by filtering unwanted packets. A packet filter is simply a boolean valued function on a packet. If the value is true the kernel copies the packet for the application and if it is false the packet is ignored. Historically there are two methods to the filter abstraction, a boolean expression tree and a directed acyclic control flow graph or CFG [41]. These two models of filtering are computationally equivalent: that is, any filter that can be expressed in terms of one can be expressed in terms of the other. However in implementation they are very different. The tree model maps into code for the stack machines while the CFG model maps into code for the register machines. Since most of the modern machines are register based machines, the CFG approach is a more efficient implementation.

BPF uses the CFG filter model. Each application that opens a BPF device can load its own filter that is then applied by BPF to each packet. One can write his own filter programs in the machine language of this pseudo machine, but the simplest is to compile ASCII strings into this machine language using the pcapcompile function of the *libpcap* library (described in later sections).

BPF Pseudo Machine

BPF basically emulates a small computer system at the kernel level. It has its own instruction set, addressing modes, Arithmetic and Logic Unit (ALU) instructions, and branch instructions. The program that the BPF executes is passed to it by *ioctl*() system calls. By executing this program, BPF filters the packets passed to it by the driver. So a read from BPF will return the packets which pass this filter. However, for the write operation there is no filtering.

BPF instructions are 64 bits long with the first 16 bits representing the opcode. The basic addressing modes include a copy of the packet data from a fixed or variable offset into a register, immediate mode and absolute memory addressing mode. However, the store instructions do not have the option of addressing packet data. This is not very surprising, because BPF is designed as a packet filter not as a packet processor. For other types of BPF instructions, including the ALU and branch instructions, we refer to [41].

The techniques used by BPF to reduce its overhead are summarized below.

- The BPF filtering is within the kernel, which minimizes the data copied from BPF to application. Otherwise, if every packet has to copied, BPF will have trouble keeping up with fast datalinks.

- Only a certain length, called capture length, of a captured packet is transferred to the application as most applications need only a packet header, but not the

data. This also reduces the data copied by the BPF to the application.

- BPF buffers collect the data destined for the application and this buffer is transferred only when the buffer is full or when the read time out expires. BPF allows the application to set this value.

- BPF maintains two buffers for each application and fills one while the other is being copied to the application. This technique is called the double buffering technique.

To access BPF one must open a BPF device that is not currently open. For example one would try BPF device '*/dev/bpf0*' and if it is busy, then try '*/dev/bpf1*', and so on. Once the device is opened, ioctl commands are used to set characteristics of the device, for example to load the filter, set the read time-out or the buffer size, attach a datalink to the BPF device, enable the promiscuous mode, etc. The input-output is then performed using normal read and write system calls. Since reading data from BPF requires the physical movement of data between kernel and user address spaces, BPF also has a time-out feature which works like this: When the buffer is not full, read system calls will first sleep a certain amount of time and then return, either because the buffer is filled while the process is sleeping *or* because the timer is expired. This feature is intended to reduce the number of kernel/user mode switches at the cost of longer response times from the BPF device.

5.3 DataLink Provider Interface (DLPI)

System V release 4 provides datalink access through DLPI, the datalink provider interface. DLPI is designed by *AT&T* to interface the services provided by the datalink layer. DLPI is a protocol independent interface and it can be accessed by sending and receiving stream messages [42].

DLPI primitives

The DLPI primitives are defined in terms of stream messages. All DLPI messages are either the *M_PROTO* or *M_PCPROTO* type and contain an appropriate DLPI message structure in the leading mblk messages [43, 45]. The DLPI primitives are generally issued in a request/acknowledge scenario, in which user issues a request message and waits for an acknowledge or error response message from the provider as shown in Figure 5.2.

Consumers of service interface can generate request primitives, while providers generate both response and event primitives. There are six classes of primitives in DLPI.

1. local management primitives enable a consumer to query and control the provider.

2. connection establishment primitives enable a consumer to create a virtual cir-

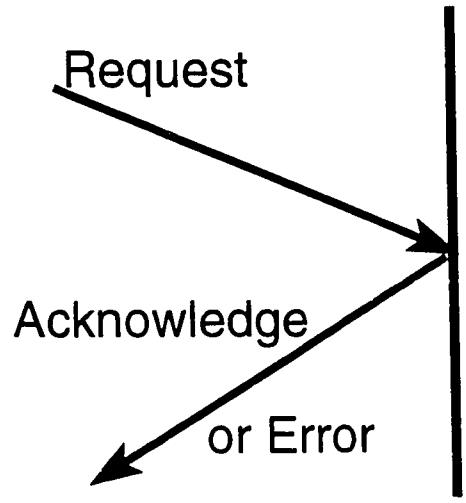


Figure 5.2: Request and acknowledgement

cuit.

3. Data transfer primitives enable a consumer to communicate with a peer DLPI consumer.

4. Reset primitives allow a datalink connection to be re-synchronized.

5. Connection release primitives allow a consumer to tear down a virtual circuit.

6. Test primitives allow a consumer to test the operation of the provider.

The following table summarizes the subset of DLPI primitives.

Primitive	Type	Description
DL_INFO_REQ	Request	Get information
DL_INFO_ACK	Response	Request information
DL_ERROR_ACK	Response	Request failed
DL_BIND_REQ	Request	Bind an address
DL_BIND_ACK	Response	Bind successful
DL_UNBIND_REQ	Request	Unbind an address
DL_OK_ACK	Response	Operation successful
DL_UNITDATA_REQ	Request	Transmit data
DL_UNITDATA_IND	Event	Data received
DL_UDERROR_IND	Event	Transmission failed

To use the DLPI conform driver, an application must first open the driver and attach the stream to the physical medium and bind a service access point (SAP) to the file descriptor. Two styles of drivers are supported in DLPI, distinguished by the way they enable the DLS user to choose a particular physical point of attachment (PPA). One type of driver attaches PPA implicitly when the driver is opened. The other type of drivers require explicit use of service interface attach primitives to select PPA. This is often used when there are more than one physical communication channels supported by the driver and an application is required to choose the channel explicitly. If there is only one physical channel of communication like in ethernet then the implicit way is used.

Each consumer of DLPI is assigned an abstraction called a datalink service access point (DLSAP). DLSAP consists of a physical address and SAP address, to represent the consumer's point of communication with the datalink provider. The DLSAP represents a communication end point except that DLPI allows providers to map a DLSAP to several communication end points (Streams). After file descriptor is attached to PPA the user must issue a datalink bind request (*DL_BIND_REQ*) primitive to associate the attached file descriptor with a full DLSAP address. After the file descriptor has been attached and bound the user can send and receive raw datalink frames or issue additional, optional features of the DLS provider.

The pfmod and bufmod STREAM modules

Pfmod and bufmod are STREAMS modules which basically do filtering and buffering at the kernel level. They are not STREAMS drivers; they are simply STREAMS modules. A user process first opens a device with a DLPI compliant driver, then pushes pfmod and bufmod modules to the stream and configures them by STREAMS messages belonging to the DLPI message set. The main function of pfmod is filtering the data going upstream but passing the data going downstream without any change. Similar to BPF, pfmod is also programmable and has its own instructions, but in contrast to BPF, it uses a boolean expression tree. For details of the pfmod instructions, we refer the reader to [44].

As in the BPF case, buffering the data before it is read by the user level process will improve the performance. However, this is not done by pfmod. A separate STREAMS module, bufmod, is used which has a time-out feature similar to BPF: When the buffer is not full, the read( ) system call will cause the user process to sleep a certain amount of time and then wake it up either because the buffer is filled while the process is sleeping *or* the time-out has expired. This feature, as in the BPF case, is intended to reduce the number of kernel/user mode switches at the cost of longer response times from the device. Normally,BPF is three to twenty times faster than boolean expression tree depending upon the complexity of the filter [41].

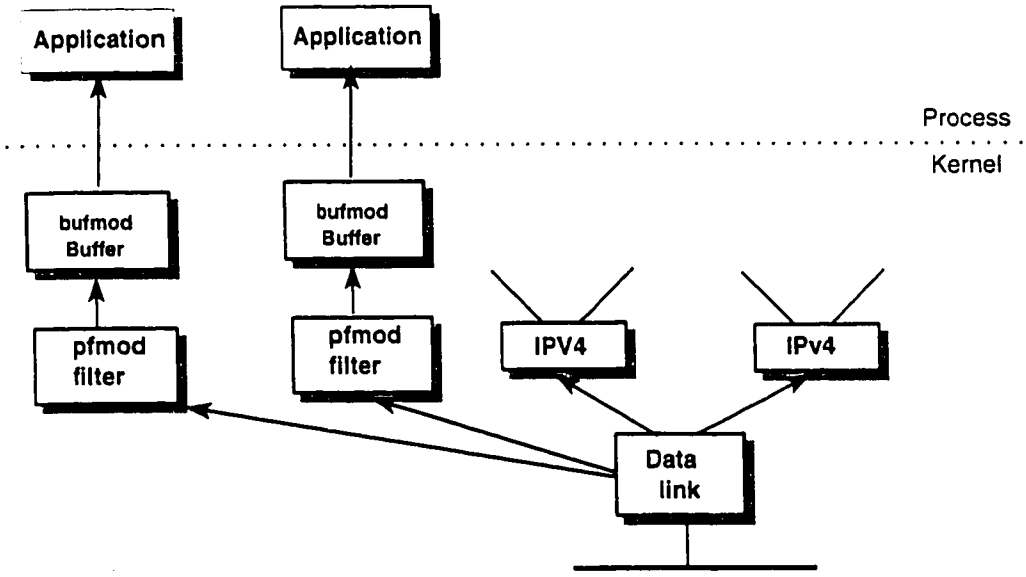


Figure 5.3: Packet capturing using DLPI

5.4 Packet Capture Library (*libpcap*)

The packet capture library, *libpcap*, provides an implementation independent access to the underlying packet capture facility provided by the operating system. Currently it supports BPF under Berkeley derived kernels, NIT under SunOS 4.1.x, the Linux *SOCK_PACKET* socket, DLPI under Solaris 2.x, and a few other operating systems. This library has twenty five functions to facilitate datalink access. Some important *libpcap* library functions are enumerated below with a brief explanation [41].

1. **Choosing the packet capture device**

   The function *pcap_lookupdev* chooses a device. It issues SIOCGIFCONF ioctl and chooses the lowest numbered device that is up. If there is an error, this function fills the error string in the supplied array. Many pcap library functions fill in an error string if an error occurs.

2. **Opening a device**

   The function *pcap_open_live* is used to obtain a packet capture descriptor to look at packets on the network. The first argument is a device name (string that specifies the network device) to open; The second argument specifies the maximum number of bytes to capture; The third argument is promiscuous flag; The fourth argument specifies the read time-out in milliseconds; The final

argument is used to return error text and is only set when this function fails and returns NULL. Instead of opening a device, one can read from a previously saved file using *pcap_open_offline*. The file name specifies the name of the file to open. The file has the same format as those used by tcpdump.

3. **Obtaining network address and subnet mask**

   The function *pcap_lookupnet* is used to determine the network address and subnet mask associated with the network device. If there is an error, NULL is returned and the error string is filled in the given array.

4. **Compiling and loading the filter**

   The function *pcap_compile* takes a string and compiles it into the compile program. This string specifies which packets should be entertained. The *pcap_setfilter* takes the filter program that is compiled and loads it into the packet capture device.

5. **Reading packets**

   The function *pcap_dispatch* is used to collect and process packets. The argument count specifies the maximum number of packets to process before returning. If the count is set to -1, then it processes all the packets received in one buffer. A count of 0 processes all packets until an error occurs (or EOF is reached). *pcap_next* returns a u_char pointer to the next packet. This function internally calls the *pcap_dispatch* function.

6. **Fetching packet capture statistics**

   The function *pcap_stats* fetches the packet capture statistics from the start of the run to the time of the call. This function tells us number of packets received by the filter and number of packets dropped by the kernel.

5.5 Libpcap+, Enhancement of *libpcap*

The *libpcap* library allows user applications only to read raw data packets from the network. It does not allow users to write their own data packets to the network. But for several purposes there is a need to read from as well as write to the network. Therefore, we have modified the *libpcap* library. The new library is named *libpcap+* [1]. The *libpcap+* library contains three extra functions, in addition to *libpcap* functions.

The added functions are:

1. pcap_write : Writes frames (for example ethernet frames) to the network.
2. pcap_set_pcapbuffer : Fills the buffer of pcap structure with the given data.
3. pcap_set_pcapbufsize : Sets the buffer size of pcap structure to the specified length.

[1] Code of this library is available in the attached disk

For sending an ethernet frame using *libpcap+*, initially we need to create an Ethernet frame in memory. The format DEC's Ethernet frame is shown in Figure 5.4. After creating an Ethernet frame in memory with the desired source address, destination address and data, it should be place at a specified position in the pcap structure. The functions *pcap_set_pcapbuffer* and *pcap_set_pcapbufsize* place the ethernet frame at appropriate position in the pcap structure. Finally the function *pcap_write* writes the ethernet frame to the network.

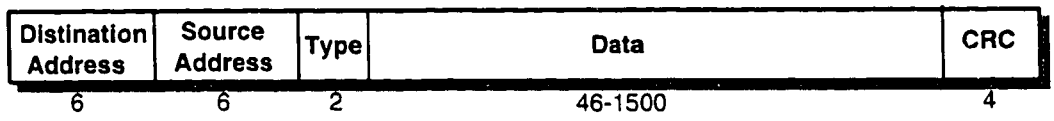


Figure 5.4: Ethernet Encapsulation (RFC 894)

5.6 Datalink Layer Interface (DLI) Library

Eventhough we have *libpcap+* library, it is not easy to use these functions. It is very difficult to remember the sequence of functions and complex data types needed for reading or writing raw data packets from the network. Even after taking the trouble, if one finishes the application, and if the application has a bug in the program, the whole system will be crashed. So, to ease the datalink access, we have added a new library called DLI, on the top of *libpcap+*. With this library, even an ordinary user can easily write applications which directly access the datalink layer. Now if there is a bug in the program only that program will be terminated, but not the entire

system. DLI, was written using Object Oriented Programming (OOP) concepts of C++. This makes a user of the DLI [2] library very comfortable.

Summary of DLI C and C++ APIs

In the DLI library, each datalink layer access point is represented by a structure struct dls which encapsulates read and write pointers, and an integer which indicates the type of the medium.

The DLI library has the following C APIs:

- struct dls dl_open(char* nidev): Generate a datalink layer access point for the network interface device nidev.
- int dl_close(struct dls Dap): Close the datalink access point corresponding to Dap.
- char *dl_read(struct dls Dap, int* len): Read from datalink layer access point corresponding to Dap. The return value is the address of the buffer which stores the read value. The length of the read data is written to len.
- int dl_write(struct dls Dap, char* buf, int len): Write to the datalink layer access point corresponding to Dap. The length of the data is len and its address is buf.

[2] Code of DLI is available in attached disk

- int dl_settimeout(struct dls Dap, int to_ms): Set the read time-out for the datalink layer access point Dap to to_ms milliseconds.

- int dl_setfilter(struct dls Dap, int pt): Set the kernel-level packet filter for the datalink layer access point Dap, so that only the packets of type pt are passed to the application.

On the top of these C APIs, we have also developed the Packet < classT > template, the PACKET class (which is basically Packet < char >) and DataLink classes. The PACKET class provides an abstraction of the packet concept, and defines methods to construct PACKET objects from character pointers and from other **PACKET** objects. The operators '+', '=', and '==' are overloaded: The '+' operator corresponds to concatenation, '=' to assignment, and '==' to comparison for objects of type PACKET. Finally operator()(int) and operator()(int, int) return **PACKET&** objects which select a certain byte or range of bytes of a PACKET object.

The DataLink class provides an abstraction of the datalink layer concept. This class has setfilter and settimeout methods for setting the kernel-level packet filter and read time-out. Furthermore, operator<<(const PACKET&) and operator>>(PACKET&) methods are defined. Their return value is DataLink&, and their main functionality is to write to or to read from the datalink layer.

5.7 An Example Procedure For Testing Congestion Control Algorithms Using DLI

In this section a simple procedure for testing congestion control algorithms is described. Before explaining actual procedure the following concepts need to be explained.

FIFO : A First In First Out (FIFO) is a special pipe device file which provides temporary buffer for two or more processes to communicate by writing data to and reading data from the buffer. The buffer associated with a FIFO file is allocated when the first process opens the file to read or write. The buffer is discarded when all the processes which are connected to the FIFO close their references. The data stored in a FIFO can be accessed in a first-in-first-out manner, hence it is called a FIFO. A FIFO file can be created in UNIX via $mkfifo$ command.

Multi-threaded Programming: A thread is a piece of program code executed in a serial fashion. In a single-threaded program only one piece of program code is executed at one time. On the other hand, a multi-threaded program can have several pieces of its code executed concurrently at any one time. Multi-threaded programming has several advantages: it improves throughput and responsiveness to users, it allows programmers to structure their code into

independently executable units and maximize concurrency, etc. A thread in UNIX can be created by using the function *pthread_create*.

Thread Synchronization : Threads in a process share the same address space as the process. This means that global and static variables in the process are accessible by all its threads. To ensure the correct manipulation of these variables, threads must use some methods to synchronize their operations. Specifically, no thread should change a variable while other threads are reading that variable value and vice versa. To solve thread synchronization problems the following objects are provided in UNIX.

- Mutually exclusive locks (mutex locks)
- Condition variables
- Semaphores

Of the above mutex locks are the most primitive and efficient to use. We have used mutex locks for thread synchronization. The *pthread_mutex_init* function can be used initialize a mutex lock in UNIX.

For testing a congestion control algorithm three main components need to be simulated namely, source, switch, and destination. Simulation of each individual component is briefly described below and corresponding code is given in Appendix A.

Sources

A source should perform the following two operations.

- It should generate data and send it to switch.
- It should receive the feedback from the switch and adjust its data rate accordingly.

Thus, each source can be simulated by a two threaded program. One thread generates and sends data to the switch and the other receives the feedback from the switch. The C++ code of a 'source program' is given in appendix A.1.

Reader and writer threads are two different classes which are inherited from a Thread class [3]. The writer class when instantiated opens a FIFO in write only mode. This FIFO acts as a forward communication channel between source and switch. Once a channel is opened, write thread continuously writes data (characters) to the FIFO with a pause between two write operations. This waiting period depends on the feedback value received from the switch. If the feedback is a large value then the waiting period is small and vice versa.

The reader class also opens a FIFO in read only mode for reading feedback sent by the switch. This feedback is used by the writer class for controlling data rates. Any number of sources can be started by starting different instances of this program with different FIFO files.

[3] Code is given in the attached disk

Switch

A switch should do the following operations.

- It should read data from the sources and keep the data in a common queue.
- It should despatch the data in the queue to the destination.
- It should process the feedback received from the destination.
- It should calculate the rates to be assigned to each source using a congestion control algorithm.

Thus, switch can also be simulated by a multi-threaded program. This program is a bit more complex than the source program. An illustrating code of this program is given appendix A.2. This program uses three independent threads for reading data from three sources. The reader class in this program do the job of reading data from a source (FIFO) and writing it to a queue. As the same queue is used by the many threads, access to the queue should be properly synchronized. The code for this queue (*MT_Queue.h*) is given in the attached disk. Since there are three sources three objects of reader class need to be instantiated. The writer class reads a character at a time from the queue and sends it to destination. Between two read operations a writer object waits for sometime. This waiting time depends on the available capacity (C) which is nothing but the received feedback from the destination. Note that we have used overloaded operators like '+' for packet concatenation,

and ' $\ll$' for writing packets to the destination, etc. These overloaded operators simplified the code.

The feeder class implements actual congestion control algorithm. The code of a simple adaptive control algorithm without tuning is implemented here. The feeder class also reads the datalink layer for the feedback sent by the destination. Note that overloaded operators '() (*int*)' and '() (*int*, *int*)' are used for getting required parts of the packet (feedback). Using available capacity and queue length the congestion control algorithm calculates the rates to be allocated to the sources and writes them to respective FIFOs as feedback.

Destination

Destination is simulated by a single threaded program. This program takes two arguments namely, host and destination address. It opens network interface card and continuously read packets addressed to this host. After every '*p*' (say 1000) it sends a packet to switch with current available capacity value. This value is obtained randomly (can be between 1 and 1000).

Testing

In the above programs DLI library is using ethernet frames for communication between switch and destination. This needs destination and switch simulation programs to run on computers which are on the same LAN segment. One can start the

simulation of the network by starting the destination program. The syntax of the command is given below.

```
Dest [1] nei0@pc-syslab ed0@pc-ftp-syslab &
```

Where pc-syslab and pc-ftp-syslab are symbolic names of the machines on which destination and switch programs are run respectively. Ed0 and nei0 are ethernet interfaces on these machines. Source programs should be started on pc-ftp-syslab. The syntax for starting a source is

```
Src srcfb1 data1 &
```

where srcfb1 and data1 are FIFOs which are used for receiving feedback and sending data respectively. Any number of sources can be started with above command but different FIFO names should be given. Switch program should also be started on pc-ftp-syslab. The syntax to start a switch simulator is

```
Switch data1 data2 data3 ed0@pc-ftp-syslab nei0@pc-syslab&
```

where data1, data2 and data3 are the FIFOs from where switch has to read data. It should be noted that the programs given in the appendix are just illustrating a procedure to test congestion control algorithms. These programs need to tuned further for better performance.

[1]Dest, Src and Switch be the executable code of destination, source and switch respectively.

Chapter 6

CONCLUSION AND RECOMMENDATIONS FOR FUTURE RESEARCH

6.1 Conclusion

In this thesis explicit rate based congestion controllers are designed for ATM's ABR service class using robust control theory. The designed controllers are H_2, H_∞, mixed H_2/H_∞ and adaptive controllers. Furthermore, these controllers are evaluated under different practical scenarios. Our experimental results show that the performance of the designed controllers is very good in terms of all the designed characteristics.

Fairness, an essential characteristic of ABR service class, is accomplished by all the designed controllers. This should not be surprising since special care was taken at the time of design itself. The adaptive controller is the fairest among the designed controllers since the same feedback is given to all the sources. Nevertheless, robust controllers are equally fair in allocating the available bandwidth.

Simulation results show that all designed controllers efficiently utilize available bandwidth except adaptive controller. Especially when the traffic conditions change dynamically, the adaptive controller's network utilization is very poor. This is because of its poor responsiveness, which is in turn due to its slow adaption process. On the other hand responsiveness of the H_∞ controller is best among all the designed controllers.

In terms of buffer requirements again adaptive controller is the least preferred controller. In ideal conditions the adaptive controller's buffer requirement is very small when compared to other controllers. But, in practice the network state hardly remains constant. Therefore, robust controllers should again be opted.

In brief we can conclude that the robust controllers nicely satisfy all the design specifications. But, these controllers are computationally very costly. On the other hand, computational overhead is very less for adaptive controller but, operating parameters need to be tuned for individual situations depending on traffic characteristics which results in poor response times.

The designed controllers are also compared with the Proportional (P) and Proportional Integrator (PI) controllers available in the literature. The obtained results show that the designed controllers clearly outperform P and PI controllers in all design aspects.

Without restricting ourselves to simulations, we have also developed a datalink access library which can be used to test and implement congestion control algorithms. The developed library supports both SVR4's DLPI and BSD's BPF. Moreover, this library is developed using OOP features of $C++$ to easy end users. Using this library, an example procedure is also given for real time simulation of congestion control algorithms.

6.2 Recommendations

First and foremost recommendation is to use more advanced control techniques like L_1 or mixed L_1/H_2 controllers. By using L_1 control, one can incorporate time domain constraints like $Q_{min} \leq Q(n) \leq Q_{max}$. But, usually L_1 controllers are of higher order, design procedure is difficult (no MATLAB toolbox is available to the best of our knowledge), and controller reduction is necessary before implementation. Another good approach is to use neural network based flow controllers. A neural network is a massively parallel distributed processing system that is suitable for storing knowledge and make it available for future use. Neural networks are thought

to provide fast, flexible, adaptive and intelligent control in ATM. The advantage of this approach is that whole neural network can be implemented in small VLSI chip. But, it takes a lot of time to train neural networks.

Since the network traffic is stochastic in nature and it is not possible to estimate traffic parameters. So one can try stochastic prediction techniques together with linear programming techniques to find an optimum flow controller. The disadvantage of this approach is that it may introduce a considerable computational overhead.

In the designed library packet processing is in the user area. This means, packet has to be transferred first to the user area before it can be processed. This results in extra overhead due to memory transfers. On the other hand, one can process packets in the kernel itself, there by reducing memory traffic, which is major bottle neck in the modern work stations.

Finally, our sincere recommendation is towards real time implementation of designed congestion controllers using DLI library.

Appendix A

Example Simulation Programs

A.1 Source Simulation Program

```
#include "Thread.h"
#include <iostream.h>
#include <stdlib.h>
#include <unistd.h>
#include <fnctl.h>
#include <sys/types.h>
// Reader Thread: reads feedback from a given FIFO
class Reader: public Thread {
private:
    int feedback, *rate;
    void* thread_code();
public:
    Reader(char&, int);
};

Reader::Reader(char& input1, int rate1) {
    feedback = open(input1, O_RDONLY, 0); rate = &rate1;
}

void* Reader::thread_code() {
     while (TRUE) {
         read(feedback, rate, sizeof(int));
         yield();
      }
}
```

```
// Writer Thread
class Writer: public Thread {
private:
    int out, *rate;
    void* thread_code();
    void* wait1(int);
public:
    Writer(char&, int);
};

Writer::Writer(char& outfile1, int rate1) {
     out= open(outfile1, O_WRONLY, 0); rate = &rate1;
}
void* Writer::thread_code() {
     char data = '7';
     int  i;
     while (TRUE) {
          for ( i =0; i< (*rate); i++){
             write(out, (void*)data, sizeof(char));
             wait1(1000/(*rate));
           }
          yield();
      }
}

void* Writer::wait1(int iter) {
      for( ; iter; iter--); // some kind of delay
}

main(int argc, char argv[][]) {
        int rate = 10;
        Reader R(argv[1], rate);
        Writer W(argv[2], rate);
        R.start(); // Reads the feedback
        W.start(); // writes the data
        while (TRUE);
}
```

A.2 Switch Simulation Program

```
#include "Thread.h"
```

```
#include "MT_Queue.h"
#include "Packet.h"
#include "Dla.h"
#include <iostream.h>
#include <stdlib.h>
#include <unistd.h>
#include <fnctl.h>
#include <sys/types.h>
// Reader Thread: reads feedback from a given FIFO
class Reader: public Thread {
private:
    int input;
    MT_Queue *Qp;
    void* thread_code();
public:
    Reader(char&, MT_Queue&);
};

Reader::Reader(char& input1, MT_Queue &Q) {
    input = open(input1, O_RDONLY, 0);
    this->Qp = &Q;
}

void* Reader::thread_code() {
    char data; // A variable to hold the feedback
     while (TRUE) {
        read(input, &data, sizeof(char));
        (*Qp) << data;
         yield();
      }
}
class Writer: public Thread {
private:
    char data;
    PACKET P;
    DataLink *DL;
    MT_Queue *Qp;
    void* thread_code();
public:
    Writer(PACKET&, MT_Queue&, DataLink& );
};
```

```
Writer::Writer(PACKET& P1, MT_Queue &Q, DataLink& D) {
    this->Qp = &Q;
    this->DL = &D;
    P = P1;
}
void* Writer::thread_code() {
    while (TRUE) {
       (*Qp) >> data;
       PACKET P_type(data, 1);
       (*DL)<< (P + P_data );
       yield();
     }
}

class Feeder: public Thread {
private:
    int src1, src2, src3, Prev_C, Prev_Q;
    MT_Queue *Qp;
    DataLink *DL;
    void* thread_code();
public:
    Feeder(MT_Queue&, DataLink&);
};

Feeder::Feeder(MT_Queue& Q, DataLink& D) {
    src1 = open("srcfb1", O_WRONLY, 0);
    src2 = open("srcfb2", O_WRONLY, 0);
    src3 = open("srcfb3", O_WRONLY, 0);
    Prev_Q = 0; Prev_C = 0;
    this->Qp = &Q;
    this->DL = &D;
}

void* Feeder::thread_code() {
    int feedback= 0; // A variable to hold the feedback
    int k1 =0, k2 = 0.333, k3 = 0.333;
    int C = 100;
    char fd[2];
    while (TRUE) {
       (*DL) >> P;
```

```
        fd[0] = P(12);    fd[1] = P(13);
       sscanf(fd,"%d", &C);
       feedback = k1 + k2(Prev_Q - (*Qp).lenght()) +k3(C- Prev_C);
       k1= feedback;
       write(src1, (void*)feedback, sizeof(int));
       write(src2, (void*)feedback, sizeof(int));
       write(src3, (void*)feedback, sizeof(int));
       yield();
     }
 }
/* The main program begins here */
main(int argc, char argv[][]) {
    char eaddr[6], if_name[20], d_eaddr[6], if_name1[20];
    char type[2] = {0xab, 0xcd};
    MT_Queue Q(100);
    PACKET P;
    process_args(argv[4], if_name, eaddr);
    process_args(argv[5], if_name1, d_eaddr);
    DataLink DL(if_name);
    DL.setfilter(0xabcd);
    PACKET P_eaddr(eaddr, 6);
    PACKET P_deaddr(d_eaddr, 6);
    PACKET P_type(type, 2);
    P = P_deaddr+P_eaddr+P_type;
    Reader S1(argv[1],Q), S2(argv[2], Q), S3(argv[3], Q);
    Writer W(P, Q, DL);
    Feeder F(Q, DL);

    S1.start(); S2.start(); S3.start(); // Reads from three sources
    W.start(); // writes the data to destination
    F.start();

    while (TRUE);
 }
```

A.3 Destination Simulation Program

```
#include "Packet.h"
#include "Dla.h"
#include <iostream.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <fnctl.h>
#include <sys/types.h>

int main(int argc char argv[][]) {
    char eaddr[6], if_name[20], d_eaddr[6], if_name1[20], C[2];
    char type[2] = {0xab, 0xcd};
    PACKET P, P1;
    int count = 0;
    process_args(argv[1], if_name, eaddr);
    process_args(argv[2], if_name1, d_eaddr);
    DataLink DL(if_name);
    DL.setfilter(0xabcd);
    PACKET P_eaddr(eaddr, 6);
    PACKET P_deaddr(d_eaddr, 6);
    PACKET P_type(type, 2);
    P = P_deaddr+P_eaddr+P_type;
    for ( ; ;) {
       DL >> P1;
       count++;
       if (count >= 100) {
         fd = rand()%1000 + 1; // get a new value for capacity
         sprintf(C,"%d",fd);
         PACKET P_data(C, 2);
         DL<< (P+P_data);
         count = 0;
        }
     }
    return (0);
}
```

Bibliography

[1] D.E. McDysan and D.L. Spohn, "ATM Theory and Application," *McGraw Hill*, 1995.

[2] W. Stalling, "ISDN and B-ISDN With Frame Relay and ATM," *Prentice Hall*, 1995.

[3] The ATM Forum, "ATM Forum Traffic Management Specifications Version 4.0," *ATM Forum/95-0013R11*, March 1996.

[4] S. kalyanaraman, R. Jain, S. Fahmy and R. Goyal, "Performance and Buffering Requirements of Internet Protocols Over ATM ABR and UBR services," *IEEE Communications Magazine*, Vol. 36(6), pp. 152-157, June 1998.

[5] L. Roberts, "Can ABR Service Replace VBR Service In ATM Networks," *Proceeding of IEEE Computer Society International Conference (COSCON)*, pp. 345-350, March 1995.

[6] A. Dagiuklas and M. Ghanbari, "Rate-Based Flow Control For Video Services In ATM Networks," *IEEE GLOBECOM'96*, Vol. 1, pp. 284-288, Nov. 1996.

[7] F. Bonomi and K. W. Fendick, "The Rate-Based Flow Control Frame Work for Available Bit Rate ATM Service," *IEEE Network Magazine*, Vol. 9, pp. 25-39, March 1996.

[8] R. Jain, "Congestion Control and Traffic Management In ATM Networks: Recent Advances and a survey," *Computer Networks and ISDN Systems*, Vol. 27, Nov 1995.

[9] M. Aida and M. Nakamura, " Real time Connection Admission Control for Multiple Service Categories," *Proceedings of ICCCN'96*, Rockville, MD, 1996.

[10] M. Aida, "Real time CAC Scheme for Multiple Service Categories Including ABR with Non-zero MCR," *Proceedings of IEEE workshop* pp. 253-262, 1997.

[11] S. Youssef, I. W. Habib and T. N. Saadawi, "A Neuro Computing Controller for Bandwidth Allocation in ATM Networks," *IEEE Journal on Selected Areas in Communication*, Vol. 15 (2), pp. 191-199, 1997.

[12] I. W. Habib and T. N. Saadawi, "Controlling Flow and Avoiding Congestion In Broadband Networks," *IEEE communication Magazine*, Vol. 29 (10), pp.46-53, Oct. 1991.

[13] W. Chen, W. Sheng, and C. Lee, "A Policing Algorithm for MPEG Stream on ATM Networks," *IEEE International Conference on Communications. Montereal* Vol. 1, pp. 545-549, March 1989.

[14] A. Tarraf, I. W. Habib, T. N. Saadawi, "A Novel Neural Network Controller Using Reinforcement Learning Method For ATM traffic Policing," *IEEE Journal on Selected Areas in Communication*, Vol. 12(6), pp. 1088-1096, Aug. 1994.

[15] M. Butto, E. Cavalleno, and A. Tonietti, "Effectiveness of Leaky Bucket Policing Mechanism in ATM Netwoks," *IEEE Journal on Selected Areas in Communication*, Vol. 9(3), pp. 335-342, April 1991.

[16] E. P. Ruthgeb, "Policing Mechanisms for ATM Networks Modelling and Performance Comparison," *Proceeding of 7th ITC seminar*, Morrisontown, NJ, Oct. 1990.

[17] F. Bonomi and K. W. Fendick, "The Rate-Based Flow Control Frame Work For Available Bit Rate ATM Service," *IEEE Network Magazine*, Vol. 9, pp. 25-31, March 1995.

[18] H. T. Kung and R. Morris, "Credit-Based Flow Control For ATM Networks," *IEEE Network Magazine*, Vol. 28, pp. 68-74, Nov. 1995.

[19] L. Roberts, "Enhanced PRCA (proportional rate control algorithm)," *ATM Forum Contribution 94-0735R1*, Aug. 1994.

[20] A. W. Barnhart, "Explicit Rate Performance Evaluations ." *ATM Forum Contribution 94-0983R1*, Oct. 1994.

[21] K. Y. Siu and H. Y. Tzeng, "Limits of performance in rate based control schemes," *ATM Forum Contribution 94-1077*, Nov. 1994.

[22] R. Jain, "A Sample Switch Algorithm Networks," *ATM Forum Contribution 95-0178*, Feb. 1995.

[23] D. H. K. Tsang, "A fast Switch Algorithm For ABR Traffic To Achieve Max-Min Fairness," *International Zurich Seminar on Digital Communications'96*, Springer, pp. 161-172, Feb. 1996.

[24] E. R. Charles, A.B. Randell and O'Halek, S., "A controller engineer's look at ATM congestion," *Computer Communications*, Vol. 19, 1996, pp. 226-234

[25] E. R.Charles and A. B. Randell, A Linear Control approach to explicit Rate Feedback in ATM Networks, IEEE 0-8106-7780-5/97, 1997, pp. 277-282.

[26] H. Ozbay, S. Kalyanaraman and A. Iftar "On Rate-based Congestion Control in High Speed Networks: Design of an H^{∞} based flow controller for single Bottleneck, An Adaptive Rate-based Congestion Control Scheme for ATM Networks," *Proceeding of the American Control Conference* Phildelphia, PA, Vol. 4, pp. 2376-2380.

[27] A. E. Echberg, "BISDN/ATM Traffic and Congestion Control," *IEEE Network Magazine*, Vol. 6, pp. 28-37, Sept/Oct. 1992.

[28] A. E. Echberg, B. T. Doshi, and R. Zoccolillo, "Controlling Congestion in BISDN/ATM: Issues and Strategies," *IEEE Communication Magazine*, Vol. 29, pp. 64-70, Sept. 1991

[29] P. Newman, "Traffic Management for ATM Local Area Networks," *IEEE Communication Magazine*, Vol. 32, pp. 44-50, Aug. 1994.

[30] D. Hong and T. Suda, "Congestion Control and Prevention in ATM Networks," *IEEE Network Magazine*, Vol.. 5, pp. 10-16, July/Aug. 1991.

[31] J. C. Doyle, P. Khargonekar, K. Glover and B. Francis, "State-space solutions to standard H_2 and H_∞ control," problems, *IEEE Transactions on Automatic Control*, Vol. 34(8):831-847, Aug. 1989.

[32] P. Apkarian and P. Gahinet, "A linear matrix inequality approach to H_∞ control," *Int. J. Robust and Nonlinear Control*, Vol. 4, 1994.

[33] P. Gahinet, A. Nemirovski, A. J. Laub and M. Chilali, "LMI Control Toolbox: For use with MATLAB," *The MATH WORKS Inc.*, 1995.

[34] A. Saberi, P. Sannuti and B. Chen, "H_2 Optimal Control," *Prentice Hall International series in Systems and Control Engineering*, 1995.

[35] K. Glover and J. C. Doyle, "State-space formulae for all stabilizing controllers that satisfy an H_∞ norm bound and relations to risk sensitivity," *Systems and Control Letters*, Vol. 11, pp.167-172, 1988.

[36] M. Dahleh, and I. J. Diaz-Bobillo., "Control of Uncertain Systems: A linear programming approach," *Prentice Hall, NewJersey* 1995.

[37] C. W. Scherer, "Multi objective H_2/H_∞ control", *IEEE Transactions on Automatic Control Systems*, Vol. 40, pp. 1054-1062, june 1995.

[38] T. Iwasaki and R. E. Skelton, "All Controllers For the General H_∞ Control Problem: LMI Existence Conditions and State-Space Formulas", *Automatica*, Vol. 30, pp. 1307-1317, 1994.

[39] W. R. Stevens, *Unix Network Programming, Networking APIs: Sockets and XTI*, Prentice Hall, 1998.

[40] Wright, G. R. and W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison Wesley, 1995.

[41] McCanne S. and V. Jacobson, "The BSD Packet Filter: A New Architectire for User-level Packet Capture," 1993 Winter Usenix Conference, January 25-29, 1993, San Diego, CA.

[42] Nuckolls, N., "How to use the STREAMS DataLink Provider Interface (DLPI)," Sun Technical Bulletin, September 1992.

[43] *Data Link Provider Interface Specification*, Unix International OSI Work Group, Revision 2.0.0, August 20, 1991.

[44] *SunOS 5.6 Answerbook2: Device Network Interfaces*, Sun Microsystems, Inc., 1997.

[45] STREAMS DLPI Specification, Sun Microsystems, part number 800-6915-01.

Vita

- Mohammad Nazeeruddin
- Born in Bhimavaram, Andra Pradesh, India on September 22, 1975
- Received Bachelor's degree in Electronics and Communication Engineering from Osmania University, Hyderabad, India in July, 1996.
- Received Honors Diploma in Software Development (HDSD) diploma from Bureau of Data Processing Systems (BDPS), Hyderabad, India in April, 1997.
- Completed Master's degree requirements at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia in May, 1999.
- Research areas include Systems & Network programming, Operating Systems and Object Oriented programming.